
L7|ESP SDK Documentation

Release 3.0.0-sdk.2

L7 Informatics

Jun 01, 2022

APPLICATION

1	Overview	1
1.1	Getting Started	1
1.2	Project Structure	4
1.3	Administration	7
1.4	Introduction	9
1.5	Content	9
1.6	Extensions	11
1.7	Testing	13
1.8	Windows Setup	23
1.9	Overview	26
1.10	Create Workflow	27
1.11	Create Test	32
1.12	Run Tests	37
1.13	Seed Content	37
1.14	Summary	38
1.15	Container Infrastructure	39
1.16	Content Installation	76
1.17	Hardware Recommendations	79
1.18	PostgreSQL Configuration	80
1.19	System Tuning	81
1.20	Backup and Disaster Recovery	86
1.21	Questions/Feedback	89

OVERVIEW

The L7|ESP SDK is a collection of software tools used for building content within L7 Informatics' Enterprise Science Platform. With the server development kit (SDK), you can do things like:

- Create a locally-running copy of L7|ESP for testing out the software.
- Develop *Workflows*, *Protocols*, *SampleTypes*, and other content for L7|ESP using programmatic definitions.
- Develop integration tests for verifying content functionality.

Having a set of tools for developing with the L7|ESP platform is paramount to leveraging what L7|ESP has to offer. This set of tools allows developers to create L7|ESP content programmatically and outside of the L7|ESP API, which drastically speeds up the development and testing cycles for customizing an L7|ESP installation.

Application Run L7|ESP locally and review the structure of a L7|ESP SDK project.

Development Discover how L7|ESP can be configured and extended to meet any process need.

Tutorial Learn how to develop and test content within L7|ESP using the L7|ESP SDK.

Deployment Guidance for running the L7|ESP platform successfully in production.

1.1 Getting Started

You will first need to install a container runtime as the L7|ESP software is distributed as an OCI-compliant container image.

1.1.1 Container Runtime

Docker Desktop or equivalent (e.g. Rancher Desktop)



A tool designed to make it easier to run container image(s) so you can focus on development, testing and deploying content to L7|ESP.

The L7|ESP SDK is delivered as a container image, and contains the following tools:

L7|ESP - Enterprise Science Platform



The entire L7|ESP server software suite is at your fingertips. Create local or remote environments for development in just a few minutes.

Git - Version Control System



Software version control system used collaboration and change tracking between developers who are working on the same L7|ESP SDK project in parallel.

Conda - Pipeline Python Environment



Python distribution and language-agnostic package manager used when writing custom pipeline scripts and integrations throughout development.

Ansible - Configuration Management Tool



IT automation tool used to prevent configuration drift when installing the same content or customizations to different L7|ESP environments.

1.1.2 Project Directory

An L7|ESP SDK project is a directory that contains all managed content and configuration for a given L7|ESP site and all its [DTAP](#) environments.

If you received a deployment bundle, you can use `tar` to extract the project directory:

```
$ tar xf deployment.tar.gz
```

To create a new L7|ESP SDK project, you can use `cruft` to copy a template repository:

```
$ pipx run cruft create git@bitbucket.org:lab7io/customerrepo_template.git
```

To clone an existing L7|ESP SDK project, you can use `git` to clone the repository:

```
$ git clone git@bitbucket.org:lab7io/acme.git
```

Note: The latter commands require access to L7 Informatics' Bitbucket account.

Throughout this documentation, we'll refer to an `acme` project in examples. `ACME Corp` is a fictitious project used during product testing and, therefore, does not represent a real entity.

1.1.3 Start Application

Typically, the server, the web UI, and the Python client will be included as a part of the L7|ESP SDK image.

Docker Compose

Most L7|ESP SDK projects use Docker's Compose plugin to manage container life-cycle. Compose is a tool for managing container configuration in a single file called `docker-compose.yml`.

Confirm Version

It's highly recommended that you ensure you are using the latest L7|ESP SDK image at any time, so you have the latest L7|ESP patches (security or otherwise).

You can confirm this by checking the first line in `Dockerfile`:

```
$ docker compose config | grep image
image: l7esp/server:3.0.0-sdk.N
```

Start

Once you've installed all prerequisites and requirements for using the L7|ESP SDK, you can start a local instance of the application using Docker Compose:

```
$ docker compose up --detach
```

This task will take a few minutes while it pulls the L7|ESP SDK container image and creates a running L7|ESP SDK container for you to develop in. If you run into any issues, please contact [L7 Informatics' Support](#).

After provisioning, you'll have a running L7|ESP instance that you can develop content for locally. For more information on configuring your application and creating content definitions, see the [Development](#) section of the documentation. For an in-depth example of creating content using the L7|ESP SDK, see the [Tutorial](#) section.

You may also check the container logs to see if there are any errors:

```
$ docker compose logs --follow
```

Note: Press `Ctrl+C` to exit the container logs.

Shell

After starting the project, you'll need to enter the L7|ESP SDK container's shell to do development and testing. All necessary tools are installed inside to container to simplify the process of getting started with the L7|ESP SDK.

To get a shell into the L7|ESP SDK container image, you can do the following:

```
$ # get a shell into the container
$ docker compose exec server bash
```

Note: At this point, you should have a Bash prompt inside the container, instead of your host machine.

Installation

Now, to install L7|ESP SDK project inside the L7|ESP SDK container running on your machine, run:

```
$ make install
```

Virtual Environments

To perform certain tasks, you may need to switch Python virtual environments. For example, when doing pipeline script development, it is often useful to enable the Python virtual environment containing the L7|ESP Python client:

```
$ source ~/data/extensions/client/bin/activate
```

Note: Once you're finished using a virtualenv, you can deactivate the environment by running the `deactivate` command.

For more information on the `virtualenv` Python program, see the [documentation](#) section.

1.2 Project Structure

The sections below detail an overview of the structure and usage patterns of the L7|ESP SDK. For information about the L7|ESP Python client, please see the [documentation](#) section. For an in-depth example of creating content using the L7|ESP SDK, see the [Tutorial](#) section.

At a high level, the L7|ESP SDK was designed to speed up and simplify the process of building content for the L7|ESP platform. It provides a set of tools that enable rapid content development and testing, and also provides a contract by which content developers, internal and external to L7 Informatics, can communicate ideas about content.

1.2.1 Files and Directories

To start understanding and using the L7|ESP SDK, let's first go over the structure of the repository and important components:

- **Makefile** - A file used primarily for administration of L7|ESP. With the `Makefile`, you can start, stop, and reset the application. You can also use it to run testing commands.
- **docker-compose.yml** - A Docker Compose configuration file that creates a containerized development environment. This file will use Docker Compose for provisioning a development system to mirror the production environment as closely as possible.
- **roles/** - Custom Ansible roles for project deployments. General purpose roles will be defined in `app/esp-content/roles`, but roles specific to each project's deployment can be included here. Also, this folder contains development (`dev.yml`) and production (`prod.yml`) playbooks to manage development/deployment environments.
- **conf/** - All L7|ESP configuration files and data sources (`lab7.conf`).
- **content/** - All content related to this project installation, including all importable configuration files and scripts.
- **extensions/** - All extension points installed in the software. These points are reserved for custom expressions and custom API endpoints.

- `patches/` – Any patches that need applied to this project installation. This can patch the core L7|ESP codebase or any configuration files.
- `tests/` – All client-side and server-side integration tests specific to a project’s workflows and configuration.

1.2.2 Organizing Content and Tests

Now that we’ve gone over some of the components at a high level, let’s dive into some detail about the internal structure of some of these subdirectories. Here’s a high-level layout of how the L7|ESP SDK directory structure might look for an L7|ESP SDK with a couple of L7|ESP *Workflows* and *Protocols* already defined:

```

sdk/
├── README.md
├── Makefile
├── docker-compose.yml
├── requirements.txt
├── pyproject.toml
├── docs/
├── conf/
│   └── lab7.conf
├── content/
│   ├── admin/
│   │   └── Users.yml
│   ├── workflows/
│   │   └── QC-Workflow.yml
│   ├── protocols/
│   │   ├── QC-Quantification.yml
│   │   └── QC-Report.yml
│   ├── pipelines/
│   │   └── QC-Report-Pipeline.yml
│   ├── tasks/
│   │   └── generate_qc_report.py
│   └── inventory/
│       ├── Sample-Types.yml
│       └── Container-Types.yml
├── extensions/
│   ├── server/
│   │   ├── expressions.py
│   │   ├── invocables.py
│   │   └── requirements.txt
│   └── client/
│       └── expressions.js
├── patches/
│   ├── conf/
│   │   └── logs.patch
│   └── esp/
│       └── sso.patch
├── roles/
│   └── container.yml
└── tests/
    ├── __init__.py
    ├── conftest.py
    └── resources/

```

(continues on next page)

(continued from previous page)

```

├── EXP001.txt
├── test_status.py
└── test_content.py

```

We'll go over details for specific files later in this documentation.

Note: How you organize your tests and content directories is up to you, but the remainder of the repository must use the above structure (the L7|ESP SDK assumes a specific file structure).

1.2.3 Notes on Specific Files

Let's go over some of the more important files in more detail. We've already covered the purpose of the `Makefile` and `docker-compose.yml` in the root directory of the repository, so let's continue with others.

Configuration

- `pyproject.toml` - A configuration file used for specifying high-level metadata about the Python project. In the `[tool.pytest.ini_options]` section, you can change the default options used by `pytest`. Usually, this file won't need to be changed, and will only need updates if you wish to adjust how certain Python tools function. Here's an example `pyproject.toml` file:

```

[tool.pytest.ini_options]
addopts = "--log-level=INFO -v -s -p no:warnings"
testpaths = [
    tests
]

```

- `conf/lab7.conf` - A file containing all L7|ESP configuration needed to run the application. For more information on this file, please see documentation for the L7|ESP platform.

Ansible Roles

- `roles/container.yml` - An Ansible playbook used for configuring the container environment with common content used for seeding the application in both development and production environments. Here's an example container playbook:

```

---
- hosts: localhost
  connection: local
  vars_prompt:
    - name: "l7esp_password"
      prompt: "Password for L7|ESP superuser: `admin@localhost`?"
      default: password
      private: yes
  roles:
    - role: l7esp_sdk
      vars:
        client_python_version: 3.9 # Valid options are 3.7, 3.8 and 3.9
        env: dev # This will allow database access from outside.

```

(continues on next page)

(continued from previous page)

```

    db_archive: true # Whether to archive the database during installs.
    seed:
      - 'roles/seed/content.yaml' # List of seed files (from `roles/packages.yaml`)
    tasks: [] # Any additional pre/post tasks you may have (optional)

```

Information about the seed section of this config is included in the *Content* section later in the documentation.

Testing

- `tests/conftest.py`, `tests/__init__.py` - Configuration files used for testing content within the L7|ESP SDK. These files generally won't need to be altered throughout L7|ESP SDK use.

1.3 Administration

1.3.1 Docker Compose + Ansible

The L7|ESP SDK relies heavily on Docker Compose for managing the container associated with running the application. For context, see the *Getting Started* section of the documentation.

1.3.2 Running the Application

As was alluded to in the documentation, you can start the application at any time using:

```
~$ docker-compose up --detach
```

For development, all content, configuration, and L7|ESP-related packages are mounted into the container. Therefore, whatever changes you make to the repo outside of the container will be reflected in the running L7|ESP instance.

The first time you run `docker-compose up --detach`, Docker Compose will try to pull the container image and subsequently create a container with all of the roles you specify in your container playbook (`roles/container.yaml`).

After the first `docker-compose up --detach`, Docker Compose will only try to start the container, and won't try to pull the image again. If you need to re-pull the container image for some reason, you can use:

```
~$ docker-compose pull
~$ docker-compose up --detach
```

These commands will re-pull the L7|ESP SDK container image, and recreate the container if required.

After you've done all of your development and you want to stop the container, you can run:

```
~$ docker-compose down
```

This command will stop all containers without removing any data volumes. If you've halted the container, you can bring the application back up using:

```
~$ docker-compose up --detach
```

If you want to completely blow away the container **and** data volumes so you can start from scratch, you can run:

```
~$ docker-compose down --volumes
~$ docker-compose up --detach
```

Project repositories are designed to quickly get developers up and running, so there's generally no negative consequences for tearing-down a container (don't be afraid!).

Note: After the application has been started and provisioned initially, other make targets, like `reset`, `reload`, and `clear`, can be used for managing updates throughout L7|ESP SDK development.

1.3.3 Resetting the Database

There are currently two ways to remove existing content in the application. The first (and fastest) of these two methods is:

```
make clear
```

When you run `make clear`, the L7|ESP SDK will use the L7|ESP Python client to drop all existing models in the application. This drop won't completely wipe the database, and all internal history for each of the models will still be in the system. `make clear` is primarily used for quickly clearing the system between tests. For a more comprehensive reset of everything in the database, you can use:

```
make reset
```

This command will take down the application, completely remove and re-migrate the database, and then bring the application back up. Just like with `make clear`, all content will be removed from the application. Although this way of resetting the database is slower than `make clear`, it's also more comprehensive in resetting the application to its default state.

1.3.4 Reloading the Application

If the application goes down or you need to restart it for some reason, you can reload it using:

```
~$ make reload
```

This command will stop and restart all services related to the L7|ESP application. If the application is not working for some reason, this command should be attempted before contacting technical support.

1.3.5 Importing Content

After the database resets, all of the initial content configured in the repository will be removed. To re-import content, you can use:

```
~$ make import
```

This command will reload any content defined as part of your development configuration in `roles/packages.yml` or `roles/dev.yml`.

1.4 Introduction

There are several types of development you can do with the L7|ESP SDK:

Content Developing and seeding configuration for content (i.e. *Workflows*, *Protocols*, *SampleTypes*, etc.).

Extensions Developing custom expressions and custom endpoints to use within content.

Testing Testing your workflows and protocols to create automated test suites.

Windows Setup Windows development environment setup instructions using WSL2

1.5 Content

While developing with the L7|ESP SDK, all content is defined within the `content/` folder of the L7|ESP SDK. That folder can be organized however you like, but a directory structure similar to the following is recommended:

```
content/
├── admin/
│   └── Users.yml
├── workflows/
│   └── QC-Workflow.yml
├── protocols/
│   ├── QC-Quantification.yml
│   └── QC-Report.yml
├── pipelines/
│   └── QC-Report-Pipeline.yml
├── tasks/
│   └── generate_qc_report.py
├── inventory/
│   ├── Sample-Types.yml
│   └── Container-Types.yml
```

In the example above, note that directories are used to separate different types of content files. Generally, most of the content added to the `content` directory will be in a YAML format that the L7|ESP Python client can import into the application. For more information about the various types of YAML config files that can be imported into the application, please see the L7|ESP Python client documentation.

1.5.1 Referencing Resources

In addition to configuration files, resources referenced in the application (i.e. scripts or artifacts used during pipeline execution) should be referenced using a special path identifier that works in both development and production. Below are useful paths for referencing custom or stock content:

- `$LAB7DATA` - The path to the application data folder, which contains the application database, pipeline artifacts, and all custom content defined in the repository.
- `$LAB7DATA/content` - Path to the `content` directory from the L7|ESP SDK. Any files you create in the `content` directory will be automatically mapped to this production directory on install/update.
- `$LAB7DATA/common` - Path to common (not project-specific) content resources. Any stock content not part of a specific project repository may be referenced via this path.

In addition to these special paths, there are several default ParamGroups you can use when running *Pipelines*:

- `param('apps', 'python')` - A reference to the version of Python installed with the application. This version of Python contains any requirements you specify in the `requirements.txt` file within the L7|ESP SDK, the L7|ESP Integrations Python module, and also the L7|ESP Python client. It's generally recommended to use this param group instead of using `python` before calling scripts.
- `param('apps', 'integration')` - A reference to the L7|ESP Integrations Python entrypoint, which is installed alongside the application for instrument integrations and other types of support.
- `param('apps', 'client')` - A reference to the L7|ESP Python client entrypoint, which might be useful for some types of activities (i.e. ingests).

For example, if we have a script in our content directory that references another file in a separate directory, we can define our *Pipeline* to reference those two files, as follows:

```
# content folder with script and resource
content/
├── pipelines/
│   └── my-custom-pipeline.yml
├── tasks/
│   ├── my_custom_pipeline.py
│   └── pipeline_config.json
```

```
# contents of content/pipeline/my-custom-pipeline.yml
My Pipeline:

tasks:
  - My Pipeline Script:
    cmd: "param('apps', 'python') $LAB7DATA/content/tasks/my_custom_pipeline.py -c
    ↪$LAB7DATA/content/tasks/pipeline_config.json"

  - My Integration Script:
    cmd: "param('apps', 'integration') instrumentsupport illumina checkindexes --
    ↪worksheet '{{sample_sheet_uuid}}'"

  - My Ingest Script:
    cmd: "param('apps', 'client') ingest my_ingest {{infile}}"
```

1.5.2 Loading Content

After developing and testing all content that needs to be in your production instance, you need to set up configuration to specify which content is loaded into L7|ESP by default. There are two components to this:

1. Create **seed** files that define which configuration to import and what types of objects they represent in the system. A **seed** file is a standard format that is used to import configuration via the L7|ESP Python client. For information about the structure of these files, see the L7|ESP Python client documentation.
2. Update your container playbook to import those **seed** files (`roles/container.yml`).

For a simple example, let's consider a `content/` folder with a *Workflow*, two *Protocols*, and a *Pipeline* that we want to go into production. We can define a **seed** file at `roles/qc.yml`, like so (you can name the seed file however you like):

```
# Contents of: roles/qc.yml

- model: Workflow
  data: $LAB7DATA/content/workflows/QC-Workflow.yml
```

(continues on next page)

(continued from previous page)

```

- model: Protocol
  data: $LAB7DATA/content/protocols/QC-Quantification.yml

- model: Protocol
  data: $LAB7DATA/content/protocols/QC-Report.yml

- model: Pipeline
  data: $LAB7DATA/content/pipelines/QC-Report-Pipeline.yml

```

In the example above, \$LAB7DATA is an environment variable referencing the L7|ESP SDK root directory. It can be used both in development and production deployments to reference the same location.

Once we've created our **seed** file, we can update our container playbook, `roles/container.yml`, to reference that seed file for the content that L7|ESP will load into the system by default:

```

---
seed:
  - '{{ sdk }}/roles/qc.yml'

```

In the file above, `{{ sdk }}` is an Ansible variable referencing the L7|ESP SDK root directory. It can be used both in development and production deployments to reference the same location.

Once you've completed all of this configuration, you can test it out using:

```
~$ make import
```

This command will use Ansible to import all of the content, which is the exact process that is used during install/update in production to import content.

1.5.3 Biobuilds

BioBuilds is a curated collection of open-source bioinformatics tools, pre-built for Linux on both Intel x86_64 and IBM POWER8 systems as well as Mac OS X.

Here we add *samtools* to the list of Biobuilds tools that will be included on install:

```

# contents of roles/container.yml
---
biobuilds:
  - bwa
  - samtools

```

1.6 Extensions

The `extensions` folder in the L7|ESP SDK allows developers to create custom expressions and endpoints in the application. Here's an example of the structure of that directory:

```

├── extensions/
│   └── server/
│       └── expressions.py

```

(continues on next page)

(continued from previous page)

```

├── invokables.py
├── requirements.txt
├── client/
├── expressions.js

```

1.6.1 Server-Side Expressions

Custom server-side expressions available in L7|ESP *Worksheets* can be created in the `extensions/server/expressions.py`. Here's an example of creating a custom expression to do a simple quantile function:

```

# contents of extensions/server/expressions.py

# functions
def quantile(array, quant):
    import numpy as np
    return np.percentile([float(x) for x in array], float(quant*100))

# export (boilerplate)
class Namespace(object):
    def __init__(self, **kwargs):
        self.__dict__.update(kwargs)

EXPRESSIONS = Namespace(context_name='all', expr_context={
    "quantile": quantile
})

```

Now, this expression can be used in a *Protocol*, like so (below is *Protocol* configuration):

```

My Protocol:
desc: Protocol that calculates the 25th quantile of all data in a column
variables:
  - Data:
    rule: numeric
  - 25th Quantile:
    rule: numeric
    value: '{{ quantile(column_values('Data'), 25) }}'

```

1.6.2 Client-Side Expressions

Note: Client-side expressions are currently in development. This section will be updated when that feature is available.

1.6.3 Custom Endpoints

Custom invokables (endpoints) available through the application can also be created in the `extensions/server/invokables.py` file. Here's an example of creating a custom endpoint to return an 'ok' status on GET request:

```
# contents of extensions/server/expressions.py

# endpoint definitions
class Ping(object):
    """
    Ping server and return OK response.
    """

    def __call__(self, agent, *args, **kwargs):
        return {'status': 'ok'}

# export (boilerplate)
INVOKABLES = {
    'ping': Ping,
}
```

Once this custom invokable has been defined, you can access the `/api/invoke/ping` URL via authenticated request:

```
>>> from esp import base
>>> base.SESSION.get('/api/invoke/ping')
{'status': 'ok'}
```

For more information about defining custom extension points within the application, please contact L7 Informatics.

1.7 Testing

1.7.1 Running Tests

As you develop content within a project repository, it's generally good practice to do test-driven development. Test creation will be covered in later sections, but for running tests, you can use two different commands:

```
~$ make test
```

This command is the first option, or you can directly invoke `pytest` for running the tests.

```
~$ pytest
```

Since `pytest` is used for running the test suite, you can use it to run specific tests or test classes in your repository. For example:

```
~$ # A single test file
~$ pytest tests/test_qc_workflow.py

~$ # All tests in a class
~$ pytest tests/test_qc_workflow.py::TestQCWorkflow

~$ # A specific test method in a class
~$ pytest tests/test_qc_workflow.py::TestQCWorkflow::test_check_qc_values
```

In addition to running tests locally, you can also run tests on a different server. To specify a different L7|ESP instance on the command line, use the `--host` and `--port` options:

```
~$ pytest --host test.l7informatics.com --port 8005
```

Note: For running tests on a public URL, you may need to connect to the application via SSL. To do so, you'll need to use the `--ssl` option and set your port to 443. For example:

```
~$ # connecting to https://test.l7informatics.com
~$ pytest --host test.l7informatics.com --port 443 --ssl
```

Here are other options useful during test execution:

```
~$ pytest -h
usage: pytest [options] [file_or_dir] [file_or_dir] [...]

positional arguments:
  file_or_dir

...

custom options:
-N, --no-import          Skip IMPORT definitions when running tests.
-C, --teardown           Teardown content after running tests.
-P, --port               Port for accessing L7|ESP.
-S, --ssl                Use SSL when connecting to L7|ESP.
-H, --host               Host for accessing L7|ESP.
-U, --email              Email for admin user.
-X, --password           Password for admin user.
```

1.7.2 Creating Tests

There are generally two types of tests you can write to verify functionality as you develop content within L7|ESP:

- **Functional Tests** - Simple tests that can be run outside of the context of testing *Workflows* and related content. Examples of this type of testing include testing connections, integration points, server-side extensions, etc.
- **Content Tests** - Tests driven by configuration that test the functionality of *Workflows* and other types of content. These types of tests are generally written using classes and L7|ESP SDK tools for automating much of the content testing.

Functional Tests

As a gentle introduction, let's write a simple test to verify that L7|ESP is connected and accepting requests. We'll need to create a test file in the `tests` directory named `test_status.py`. The contents of that file should look something like:

```
# -*- coding: utf-8 -*-

import pytest
```

(continues on next page)

(continued from previous page)

```
def test_status():
    import esp
    running = esp.status()
    assert running, 'Could not connect to L7|ESP via python client!'
```

After creating tests, you can run them using `pytest` (this example uses the test defined above):

```
~$ pytest tests/test_status.py b
===== test session starts.
<-----
platform darwin -- Python 3.7.1, pytest-3.10.0, py-1.7.0, pluggy-0.8.0 -- /usr/local/opt/
python/bin/python3.7
cachedir: .pytest_cache
collected 1 item

tests/test_status.py::test_status Connection established!

PASSED

===== 1 passed in 0.68 seconds.
<-----
```

If L7|ESP was not running and the tests failed, you'd see the following:

```
~$ pytest tests/test_status.py b
===== test session starts.
<-----
platform darwin -- Python 3.7.1, pytest-3.10.0, py-1.7.0, pluggy-0.8.0 -- /usr/local/opt/
python/bin/python3.7
cachedir: .pytest_cache
collected 1 item

tests/test_status.py::test_status FAILED

===== 1 failed in 0.68 seconds.
<-----
```

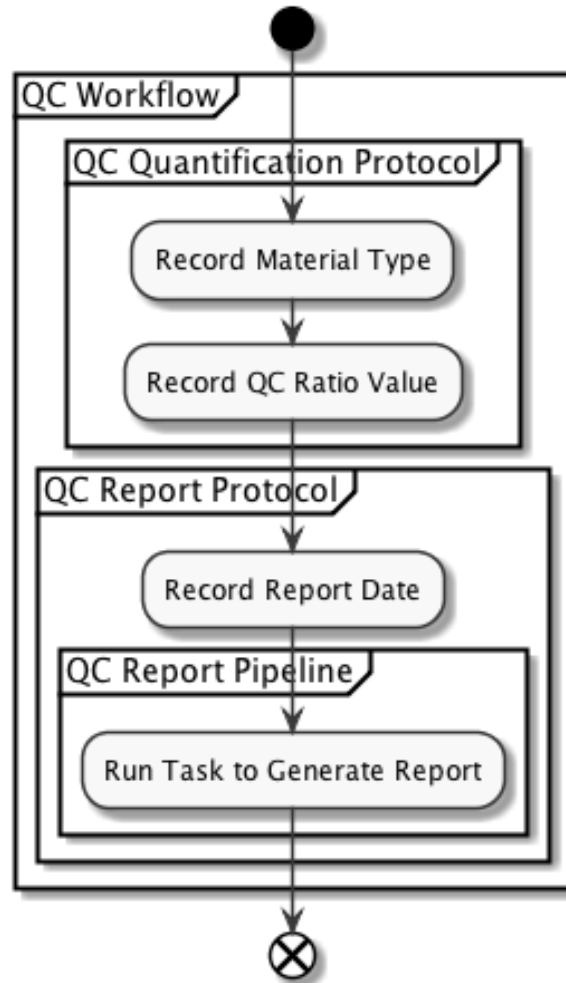
Now that we've created a test for some specific functionality in the application, let's talk about how to create tests for custom content defined in the L7|ESP SDK.

Content Tests

Along with functional testing, we can leverage L7|ESP SDK tools to help integration testing for content defined in the L7|ESP SDK. By now, you're familiar with the config-style format for defining content, so let's talk about defining tests to verify the functionality of a simple QC Workflow. For this example, our *Workflow* will have two parts:

1. A *Protocol* to capture QC metadata about a *Sample*.
2. A *Protocol* to generate a *Report* using a *Pipeline*.

Here's a schematic detailing the components of that *Workflow* at a high level:



Now that we know what type of content we're going to create, let's make some content files. First, we need to create our *Workflow*:

```

# Contents of: content/workflows/QC-Workflow.yml

QC Measurement Workflow:
  desc: Workflow to measure QC values and generate a QC report.

  tags:
    - quickstart
    - qc

  protocols:
    - QC Quantification:
      protocol: standard
      variables:
        - Type:
          rule: dropdown
          dropdown:
            - 'DNA'
            - 'RNA'
  
```

(continues on next page)

(continued from previous page)

```

- Ratio:
  rule: numeric

- QC Report:
  protocol: pipeline
  pipeline: QC Report
  variables:
    - material:
      rule: text
      value: "{{ column_value('Type', 'QC Quantification') }}"
      visible: false
    - ratio:
      rule: numeric
      value: "{{ column_value('Ratio', 'QC Quantification') }}"
      visible: false
    - Report Date:
      rule: date

```

And we also need to create the *Pipeline* that our *Workflow* references:

```

# Contents of: content/pipelines/QC-Report.yml

QC Report:
  report:
    name: Result Report
    elements:
      - - depends:
          - {file: Result Report, tasknumber: 1}
          type: raw_file
      - []
    tasks:
      - Generate QC Report:
          desc: Task to generate QC report from workflow metadata
          cmd: |+
            # Simple Task that determines if the specified 'ratio' is in the proper range

            RNA_MIN=1.8
            RNA_MAX=2.1

            DNA_MIN=1.7
            DNA_MAX=2.0

            TYPE="{{ material }}"
            RATIO="{{ ratio }}"

            if [ $TYPE = "RNA" ]; then
              PASS=`echo "$RNA_MIN <= $RATIO && $RATIO <= $RNA_MAX" | bc`;
            elif [ $TYPE = 'DNA' ]; then
              PASS=`echo "$DNA_MIN <= $RATIO && $RATIO <= $DNA_MAX" | bc`;
            fi

            echo $PASS

```

(continues on next page)

(continued from previous page)

```

        if [[ $PASS = 1 ]]; then
            echo "<b>Your sample <font color='green'>Passed</font> and contains pure
↳$TYPE</b>" >> result.html
        else
            echo "<b>Your sample <font color='red'>Failed</font> and is NOT pure $TYPE</
↳b>" >> result.html
        fi
    files:
    - Result Report:
      file_type: html
      filename_template: "{{ 'result.html' }}"

```

Now that we have our content defined, we can use L7|ESP SDK tools to write a simple test that will import these files and verify that they exist in the system. This test also enables us to verify that there are no syntax errors in our *Workflow* definitions. Our test file for doing so can look like the following, `tests/test_qc_workflow.py`:

```

# -*- coding: utf-8 -*-

# imports
import os
import unittest

from . import CONFIG, RESOURCES, CONTENT
from esp.testing import ModelLoaderMixin

# tests
class TestQCWorkflows(ModelLoaderMixin, unittest.TestCase):
    IMPORT = dict(
        Workflow=[
            os.path.join(CONTENT, 'workflows', 'QC-Workflow.yml'),
        ],
        Pipeline=[
            os.path.join(CONTENT, 'pipelines', 'QC-Report-Pipeline.yml'),
        ]
    )

```

Running this test will result in the following output:

```

~$ pytest tests/test_qc_workflow.py

===== test session starts.
↳=====
platform darwin -- Python 3.7.1, pytest-4.1.0, py-1.7.0, pluggy-0.8.0 -- /usr/local/opt/
↳python/bin/python3.7
cachedir: .pytest_cache
collected 1 item

tests/test_qc_workflow.py::TestQCWorkflows::test__content
INFO:root:Clearing existing content from database.
INFO:root:Creating Task: Generate QC Report
INFO:root:Creating Pipeline: QC Report Pipeline

```

(continues on next page)

(continued from previous page)

```

INFO:root:Creating PipelineReport: QC Report
INFO:root:Creating Protocol: QC Quantification
INFO:root:Creating Protocol: QC Report
INFO:root:Creating Workflow: QC Workflow
Successfully imported config data.
PASSED

===== 1 passed in 3.47 seconds.
↔=====

```

Along with simply testing *Workflow* definitions, we can also test that the *Workflow* works properly when used in an *Experiment*. To do this test, we can use the same format as above, but add a `DATA` attribute for what *Projects/Experiments* need to be created. For this example, we'll need to create a *Project* and two *Experiments* for testing out the *Workflow*. Here are config files for our two *Experiments*:

```
# Contents of tests/resources/QC-Test-1.yml
```

```

My QC Experiment 1:
  submit: True
  project: QC Project
  workflow: QC Workflow

  samples:
    - ESP001
    - ESP002

  protocols:
    - QC Quantification:
      data:
        ESP001:
          Type: DNA
          Ratio: 1.8
        ESP002:
          Type: RNA
          Ratio: 1.9

```

```
# Contents of tests/resources/QC-Test-2.yml
```

```

My QC Experiment 2:
  submit: True
  project: QC Project
  workflow: QC Workflow

  samples:
    - ESP003
    - ESP004

  protocols:
    - QC Quantification:
      data:
        ESP003:
          Type: DNA

```

(continues on next page)

(continued from previous page)

```

    Ratio: 1.7
ESP004:
    Type: RNA
    Ratio: 2.0

```

After creating these two configs, we can update our tests to run them after *Workflows* are created. Here is an updated version of our original test file:

```

# -*- coding: utf-8 -*-

# imports
import os
import unittest

from . import CONFIG, RESOURCES, CONTENT
from esp.testing import ModelLoaderMixin

# tests
class TestQCWorkflows(ModelLoaderMixin, unittest.TestCase):
    IMPORT = dict(
        Workflow=[
            os.path.join(CONTENT, 'workflows', 'QC-Workflow.yml'),
        ],
        Pipeline=[
            os.path.join(CONTENT, 'pipelines', 'QC-Report-Pipeline.yml'),
        ]
    )
    DATA = dict(
        Project=[
            {'name': 'QC Project'}
        ],
        Experiment=[
            os.path.join(RESOURCES, 'QC-Test-1.yml'),
            os.path.join(RESOURCES, 'QC-Test-2.yml'),
        ]
    )

```

Let's go over the components of this test class in more detail:

- `ModelLoaderMixin` - A test mixin used for managing content imports and setup/teardown during testing. For testing out content in this way, you'll want to include both `esp.testing.ModelLoaderMixin` and `unittest.TestCase` in your tests.
- `IMPORT` - A class variable that is a dictionary of mappings from model type to models that should be imported as a part of the tests. For information on the types of models you can create configuration for, see documentation for the Python client.
- `DATA` - A class variable that is similar to `IMPORT`, but is reserved specifically for creating *Experiments* and *Projects* for testing out content.

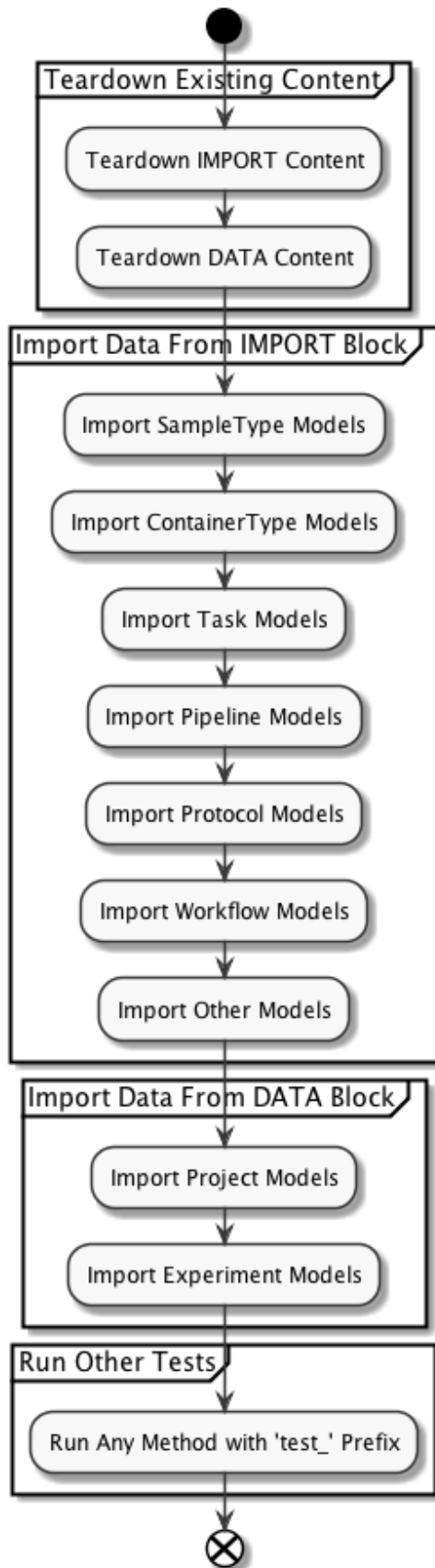
Under the hood, any content defined within the `IMPORT` and `DATA` blocks of the tests will be using the Python client to create the content by calling `Model.create()` for a specific model. For example, if your tests look like:


```
class TestQCWorkflows(ModelLoaderMixin, unittest.TestCase):
    IMPORT = dict(
        SampleType=[
            os.path.join(RESOURCES, 'inventory', 'Sample-Types.yml'),
            {
                'name': 'My Sample Type',
                'desc': 'Description for My Sample Type'
            }
        ]
    )
```

The configuration will be created with this equivalent code:

```
from esp.models import SampleType
SampleType.create(
    os.path.join(RESOURCES, 'inventory', 'Sample-Types.yml')
)
SampleType.create(
    name='My Sample Type',
    desc='Description for My Sample Type'
)
```

For additional context, the following flow diagram describes testing steps using the format described above (i.e. any test using a `ModelLoaderMixin`):



1.7.3 Using Configuration

In addition to explicitly defining what configuration you want to include in your testing, you can also import default configuration, specified in **seed** files within the repository (see the *Content* section for more information on those seed files).

To import a seed file in your testing, you can update your test class to look like the following (keeping with the example used above):

```
class TestQCWorkflows(ModelLoaderMixin, unittest.TestCase):
    SEED = os.path.join(ROLES, 'demos', 'content.yml')
    DATA = dict(
        Project=[
            {'name': 'QC Project'}
        ],
        Experiment=[
            os.path.join(RESOURCES, 'QC001.yml'),
            os.path.join(RESOURCES, 'QC002.yml'),
        ]
    )
```

1.8 Windows Setup

1.8.1 Installation Prerequisites

- Linux on Windows with WSL
 - <https://docs.microsoft.com/en-us/windows/wsl/install>
- Ubuntu v20.04-LTS (download and install from Microsoft store app)
 - <https://www.microsoft.com/store/productId/9MTTCL66CPXJ>
- Windows Terminal (download from Microsoft store app)
 - <https://www.microsoft.com/store/productId/9N0DX20HK701>
- Docker Desktop
 - Install Desktop Docker with WSL2 backend
 - <https://docs.docker.com/desktop/windows/wsl/>
- VS Code (Remote-wsl and python extensions are helpful)
 - Install VS Code for your Windows version: <https://code.visualstudio.com/>
 - Then from within the Extensions install the following extensions:
 - remote-wsl (<https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl>)
 - remote-containers (<https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-containers>)
 - python (<https://marketplace.visualstudio.com/items?itemName=ms-python.python>)

1.8.2 Setup Instructions

1. Run (or Setup for first time) Ubuntu in Windows.

Note: Windows may not be ready to install a Linux environment & will inform you to activate Windows Subsystem for Linux first. If you get such an error, see (<https://docs.microsoft.com/en-us/windows/wsl/install>) for additional details (and follow these instructions):

- Press Win+R
- Type: cmd.exe
- To gain elevated admin privileges, press: Ctrl+Shift+Enter
- At the command line, type: cd C:\WINDOWS\system32
- Then: wsl --install -d Ubuntu-20.04

-
2. Next follow the prompts to install the actual Ubuntu system.

3. Launch Ubuntu and set your username and password

- Close Ubuntu window

4. Launch windows terminal:

- In the Windows Terminal settings:
 - In “Startup” settings: Set the Default System to “ubuntu 20.04” (and save changes)
 - Under “Profiles”, change the starting directory for “Ubuntu 20.04” to:
 - * \\wsl\$\Ubuntu-20.04\home\

Note: Replace “<username>” with the username you created during the Ubuntu Install.

5. Update/install the following packages in your windows terminal linked to Ubuntu

```
sudo apt update -y && sudo apt upgrade -y
sudo apt install -y python3-pip python3-venv nano
```

6. After Ubuntu is set up, add a WSL configuration file in the /etc folder of your Ubuntu instance so it mounts the drive with metadata and the SSH works (do these with windows terminal)

```
sudo nano /etc/wsl.conf
# Type in this and save the file:
[automount]
options = "metadata"
```

7. Add a WSL configuration file in the Windows user directory (change to your username), to limit memory usage (by default WSL will consume 50% or 8GB of memory or whichever amount is smaller)

- Add Global memory limits for the Windows Environment via Powershell to your Windows home directory (e.g. C:\Users\\.wslconfig):

```
sudo nano /mnt/c/Users/<user>/.wslconfig
# Type in this and save the file:
[wsl2]
```

(continues on next page)

(continued from previous page)

```
processors=2 # Limits WSL2 VM use to two virtual processors
memory=6GB # Limits WSL2 VM memory to 6GB
```

- Capability to add memory limits on the installed distribution (Ubuntu) Environment. ↪ are not necessary to perform, but is referenced here **for** awareness:
- Access the ``/etc`` directory by entering:

```
cd /etc
# Verify you have a wsl.conf file in the /etc/ directory contents & then edit:
ls -la | grep wsl.conf
nano wsl.conf
```

Note: To see which options you can place into the configuration file, view this article: - <https://docs.microsoft.com/en-us/windows/wsl/wsl-config#configure-global-options-with-wslconfig>

1. Run Docker Desktop, enable integration with WSL2 Ubuntu & link your docker to Bitbucket.

- In Docker Desktop you will need to add the Ubuntu distribution to your WSL2 settings for WSL integration:
 - Go into “Resources” and select sub-menu “WSL Integration”: Enable integration with additional distros...
 - Turn on switch for: “Ubuntu-20.04” (first make sure “Enable integration with my default WSL distro is checked”)

If ‘Ubuntu-20.04’ switch is not present, trouble shoot using the following:

- In Powershell, type in the following command to check which version of Ubuntu you are running:

```
wsl --list -verbose
# Version should be 2 for all names.
NAME                STATE      VERSION
Ubuntu-20.04        Running    2
docker-desktop      Running    2
docker-desktop-data Running    2

# If Ubuntu-18.04 is in Version 1, enter the following command:
wsl --set-default-version 2
# Check again which version of Ubuntu you are running.
```

If you are running version 2, Refresh Docker Desktop and the Ubuntu-20.04 switch should be visible.

1.9 Overview

Below, you can find code-heavy examples of how to use the L7|ESP SDK. These examples are not meant to be comprehensive, but rather give users a feel for how to best leverage tools in the L7|ESP SDK.

It is highly recommended that readers of this document complete the L7|ESP Quick Start tutorial through the UI and Python client (see core and client L7|ESP documentation) before completing this tutorial via the L7|ESP SDK. This tutorial mirrors building out that content, so users should come in with an understanding of how content definitions work within the context of L7|ESP.

In addition, this documentation refers heavily to the L7|ESP Python client. Please see that documentation for information on usage patterns.

1.9.1 Tasks

To demonstrate building out content for a specific lab, we'll build a simple end-to-end *Workflow*. Upon completion of this example, we'll have used L7|ESP to do the following:

- Create a simple *Protocol* that tracks values associated with *Samples*.
- Create a simple *Pipeline* that generates a *Report* for each *Sample*.
- Create a *Workflow* that allows the *Protocol* and *Pipeline* to be used together for processing *Samples*.
- Organize content in the L7|ESP SDK so that it can be referenced during development.
- Create tests for evaluating functionality of that *Workflow*.
- Update the L7|ESP SDK configuration to automatically load that content on install.

In summary, at the end of this tutorial, we will have a fully functional laboratory management system with one *Workflow* already set up!

Note: L7|ESP uses many common terms for elements within the platform. Throughout this manual, terms with specific meanings in L7|ESP will always be capitalized, while lowercase versions of the same words are used in the general sense. For example, a laboratory protocol can be captured in an L7|ESP *Protocol*, and then grouped into an L7|ESP *Workflow*.

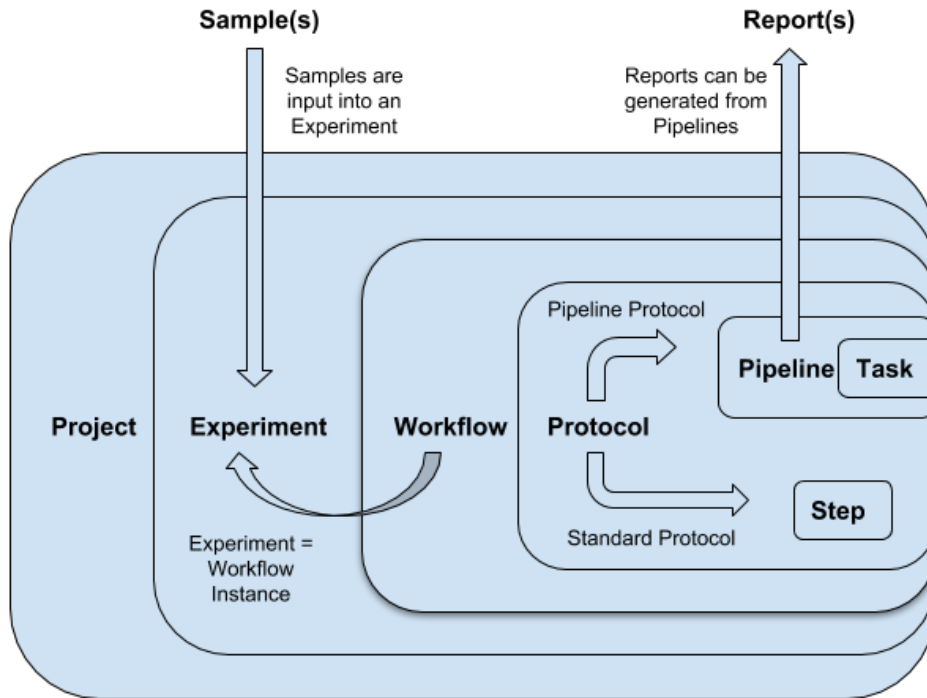
1.9.2 Terminology

While this list is not entirely comprehensive, it covers basic terminology for the scope of this introductory demo.

- **User:** Any person who uses L7|ESP
- **File:** A reference to a file registered as a Resource and its associated metadata
- **Sample:** A physical sample registered within L7|ESP. *Note that this term has a general purpose, and can be used to represent a raw sample, an aliquot, etc.*
- **Step:** A procedural step defined within an L7|ESP *Protocol*
- **Task:** A computational step in an L7|ESP *Pipeline*
- **Protocol:** A sequence of steps performed on a *Sample*. Note that this guide has the user create both standard and pipeline *Protocols*. Pipeline *Protocols* allow users to run *Pipelines* as part of a larger *Sample*-processing *Workflow*.
- **Pipeline:** An ordered collection of one or more L7|ESP *Tasks*

- **Report:** A collection of visual elements that display results from an L7|ESP *Pipeline*
- **Workflow:** An ordered collection of L7|ESP *Protocols*
- **Experiment:** An instance of a particular L7|ESP *Workflow* with associated *Samples* and data
- **Project:** A grouping mechanism for L7|ESP *Experiments, Pipelines, Samples, Reports, etc.*

A high-level diagram that displays the general relationship between these terms can be viewed below:



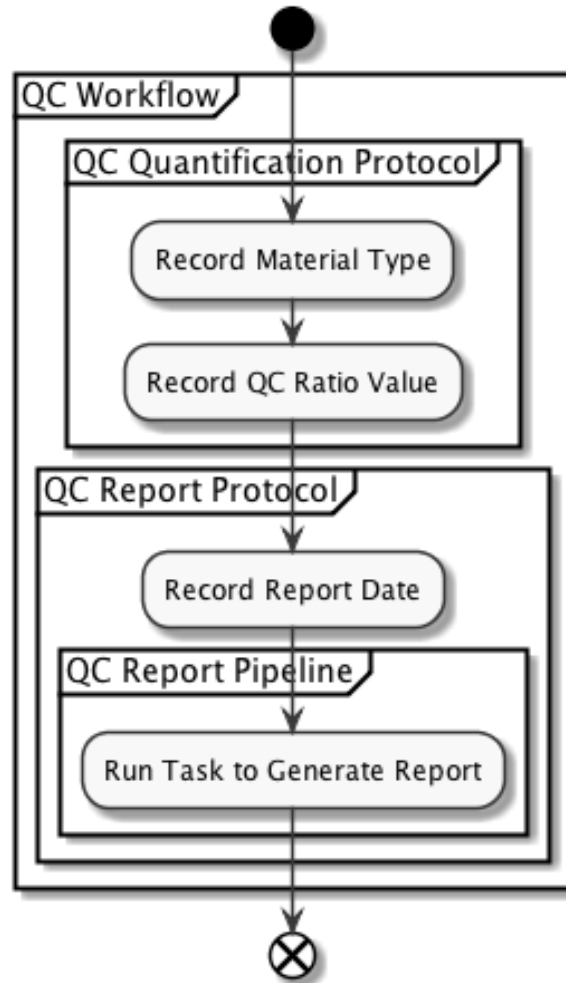
1.10 Create Workflow

For this tutorial, the user will build a simple QC *Workflow* that records a QC value and calculates a Pass/Fail result as a *Report*. By this point, you've already seen this *Workflow* in other tutorials, so we'll focus on how to create the content in the context of the L7|ESP SDK.

For this demo, our *Workflow* will have two parts:

1. A *Protocol* to capture QC metadata about a *Sample*.
2. A *Protocol* to generate a *Report* using a *Pipeline*.

Here's a schematic detailing the components of that *Protocol* at a high level:



All content developed within the L7|ESP SDK is defined within YAML config files that live in the `content` directory within the repository. For more information about the structure of the repository, see the [Project Structure](#) section of the documentation.

1.10.1 Step 1: Create a *Protocol*

Now that we know what type of content we're going to create, let's make some content files. First, we need to create our first *Protocol* (QC Quantification):

Using the L7|ESP SDK, we can create this configuration via YAML config file. For more information on the YAML config format or types of content that can be created with the YAML config format, please see the L7|ESP Python client documentation.

For this first *Protocol*, create the following file:

```

# Contents of: content/protocols/QC-Quantification.yml

QC Quantification:
  protocol: standard
  variables:
    - Type:

```

(continues on next page)

(continued from previous page)

```

    rule: dropdown
    dropdown:
      - 'DNA'
      - 'RNA'
  - Ratio:
    rule: numeric

```

In this file, we've defined an L7|ESP *Protocol* that has two columns the user will input when filling out *Worksheets: Type* and *Ratio*. *Type* is a column of dropdown type with options *DNA* and *RNA*, and *Ratio* is a numeric column. Other types of *Protocol* columns include (also listed in the L7|ESP Python client documentation):

- `string` - Column with text or string data.
- `numeric` - Column with numeric data.
- `dropdown` - Column with a dropdown selector. The options for the dropdown are specified as a list in the dropdown property.
- `checkbox` - Column with checkbox indicating yes/no status.
- `date` - Column with date picker for selecting calendar date.
- `attachment` - Column for uploading file attachment.
- `link` - Column with data containing external link.
- `barcode` - Column for scanning barcode data and printing various types of barcodes (QR, 1D, Mini DataMatrix, etc.).

Note: This list may not be fully comprehensive. For the full list, please see the L7|ESP Python client documentation.

Along with this *Protocol*, we're going to define an L7|ESP *Pipeline* and pipeline *Protocol* to include in our *Workflow*.

1.10.2 Step 2: Create a *Pipeline* to generate a *Report*:

To create a *Pipeline* for use in a *Workflow*, we need to create two things:

1. A *Pipeline* that will generate a *Report* when given *Type* and *Ratio* QC data.
2. A pipeline *Protocol* that will link the values from the *QC Quantification Protocol* to our *Pipeline*.

To start, **define the report-generating Pipeline:**

```

# Contents of: content/pipelines/QC-Report-Pipeline.yml

QC Report Pipeline:
report:
  name: Result Report
  elements:
    - depends:
      - {file: Result Report, tasknumber: 1}
      type: raw_file
    - []
tasks:
  - Generate QC Report:
    desc: Task to generate QC report from workflow metadata

```

(continues on next page)

(continued from previous page)

```

cmd: |+
  # Simple Task that determines if the specified 'ratio' is in the proper range

  RNA_MIN=1.8
  RNA_MAX=2.1

  DNA_MIN=1.7
  DNA_MAX=2.0

  TYPE="{{ material }}"
  RATIO="{{ ratio }}"

  if [ $TYPE = "RNA" ]; then
    PASS=`echo "$RNA_MIN <= $RATIO && $RATIO <= $RNA_MAX" | bc`;
  elif [ $TYPE = 'DNA' ]; then
    PASS=`echo "$DNA_MIN <= $RATIO && $RATIO <= $DNA_MAX" | bc`;
  fi

  echo $PASS

  if [[ $PASS = 1 ]]; then
    echo "<b>Your sample <font color='green'>Passed</font> and contains pure
↪$TYPE</b>" >> result.html
  else
    echo "<b>Your sample <font color='red'>Failed</font> and is NOT pure $TYPE</
↪b>" >> result.html
  fi
files:
- Result Report:
  file_type: html
  filename_template: "{{ 'result.html' }}"

```

In the block above, we're defining a *Pipeline* with a single *Task*. In that *Task*, the `cmd` block contains code that will be run as part of the *Pipeline*. This code can either be bash code or a call to an external script that will run the *Pipeline*. Also in the `cmd` block are references to `{{ material }}` and `{{ ratio }}` variables that are passed into the *Pipeline*. Next, we'll create configuration for a *Protocol* that can run this *Pipeline* as part of our *Workflow*.

1.10.3 Step 3: Create a 'pipeline' *Protocol* to record data and run the *Pipeline*:

Now, let's create our pipeline *Protocol* for running the *Pipeline* that we defined above. For this *Protocol*, we want to capture column data (Type and Ratio) from a previous *Protocol* so they can be passed as arguments (`material` and `ratio`) to the *Pipeline*. We also want to include a date column for users to specify a date when the QC data were taken.

Create the following file to capture all of this information:

```

# Contents of: content/protocols/QC-Report.yml

QC Report:
  protocol: pipeline
  pipeline: QC Report Pipeline
  variables:
    - material:

```

(continues on next page)

(continued from previous page)

```

    rule: text
    value: "{{ column_value('Type', 'QC Quantification') }}"
    visible: false
  - ratio:
    rule: numeric
    value: "{{ column_value('Ratio', 'QC Quantification') }}"
    visible: false
  - Report Date:
    rule: date

```

Above, we can reference our *Pipeline* using the `pipeline` configuration option. Additionally, for *Protocols* that are executing a *Pipeline*, `protocol: pipeline` must be explicitly specified as part of the *Protocol* definition. Finally, to carry data over from a previous *Protocol*, we're using an L7|ESP expression in the `value` field for the column. For more information on expressions and how they're used in L7|ESP, see the L7|ESP documentation.

1.10.4 Step 4: Put it all together in a *Workflow*:

Now that we our *Pipeline* and *Protocols* defined, we can put them all together in a *Workflow* that can be used in the lab.

Define the *Workflow* as follows:

```

# Contents of: content/workflows/QC-Workflow.yml

QC Workflow:
  desc: Workflow to measure QC values and generate a QC report.
  tags:
    - quickstart
    - qc

  protocols:
    - QC Quantification
    - QC Report

```

Above, since our *Protocol* configuration is already defined, we can simply reference them in the `protocols` block of the *Workflow*.

Notes

You can also define all of these items in a nested way.

For instance, if you want to define the *QC Workflow* with a single *Protocol* all in the same config file, **instead, define the *Workflow* this way:**

```

# workflow
QC Workflow:
  desc: Workflow to measure QC values and generate a QC report.
  tags:
    - quickstart
    - qc

# nested protocols
protocols:

```

(continues on next page)

(continued from previous page)

- **QC Quantification:**
 - protocol:** pipeline
 - variables:**
 - **Type:**
 - rule:** dropdown
 - dropdown:**
 - 'DNA'
 - 'RNA'
 - **Ratio:**
 - rule:** numeric

1.11 Create Test

Test-driven development is highly recommended as good practice when using the L7|ESP SDK, so we'll start our tutorial with testing. As a gentle introduction, let's write a simple test to verify that L7|ESP is connected and accepting requests.

To do this task, **create a test file in the ``tests`` directory named ``test_status.py``**. The contents of that file should look something like:

```
# -*- coding: utf-8 -*-

import pytest

def test_status():
    import esp
    running = esp.status()
    assert running, 'Could not connect to L7|ESP via Python client!'
```

Now that you've written a test, **run it using ``pytest``**:

```
~$ pytest tests/test_status.py b
===== test session starts.
<-=====
platform darwin -- Python 3.7.1, pytest-3.10.0, py-1.7.0, pluggy-0.8.0 -- /usr/local/opt/
<-python/bin/python3.7
cachedir: .pytest_cache
collected 1 item

tests/test_status.py::test_status Connection established!

PASSED

===== 1 passed in 0.68 seconds.
<-=====
```

If L7|ESP was not running and the tests failed, you'd see the following:

```
~$ pytest tests/test_status.py b
===== test session starts.
<-=====
```

(continues on next page)

(continued from previous page)

```
platform darwin -- Python 3.7.1, pytest-3.10.0, py-1.7.0, pluggy-0.8.0 -- /usr/local/opt/
↳python/bin/python3.7
cachedir: .pytest_cache
collected 1 item

tests/test_status.py::test_status FAILED

===== 1 failed in 0.68 seconds_
↳=====
```

Later in this tutorial, we'll talk about different ways of structuring tests for various types of content definitions.

1.11.1 Step 1: Create test file for importing data

Now that we have our content defined, we can use L7|ESP SDK tools to write a simple test that will import these files and verify that they exist in the system. This test also enables us to verify that there are no syntax errors in our *Workflow* definitions.

Create the test file, `tests/test_qc_workflow.py`, to look like the following:

```
# -*- coding: utf-8 -*-

# imports
import os
import unittest

from . import CONFIG, RESOURCES, CONTENT
from esp.testing import ModelLoaderMixin

# tests
class TestQCWorkflows(ModelLoaderMixin, unittest.TestCase):
    IMPORT = dict(
        Workflow=[
            os.path.join(CONTENT, 'workflows', 'QC-Workflow.yml'),
        ],
        Protocol=[
            os.path.join(CONTENT, 'protocols', 'QC-Quantification.yml'),
            os.path.join(CONTENT, 'protocols', 'QC-Report.yml'),
        ],
        Pipeline=[
            os.path.join(CONTENT, 'pipelines', 'QC-Report-Pipeline.yml'),
        ]
    )
```

Run this test to get the following output:

```
~$ pytest tests/test_qc_workflow.py

===== test session starts_
↳=====
platform darwin -- Python 3.7.1, pytest-4.1.0, py-1.7.0, pluggy-0.8.0 -- /usr/local/opt/
↳python/bin/python3.7
```

(continues on next page)

(continued from previous page)

```

cachedir: .pytest_cache
collected 1 item

tests/test_qc_workflow.py::TestQCWorkflows::test__content
INFO:root:Clearing existing content from database.
INFO:root:Creating Task: Generate QC Report
INFO:root:Creating Pipeline: QC Report Pipeline
INFO:root:Creating PipelineReport: QC Report
INFO:root:Creating Protocol: QC Quantification
INFO:root:Creating Protocol: QC Report
INFO:root:Creating Workflow: QC Workflow
Successfully imported config data.
PASSED

===== 1 passed in 3.47 seconds.
->=====

```

From this test output, we can see that all of our content was imported correctly, and references within the files were properly defined.

1.11.2 Step 2: Create *Experiment* Configs for Testing

Along with simply testing *Workflow* definitions, we can also test that the *Workflow* works properly when used in an *Experiment*. To do this, we can use the same format as above, but add a *DATA* attribute to our test class for what *Projects* and *Experiments* need to be created. For this example, we'll need to create a *Project* and two *Experiments* for testing out the *Workflow*.

Create the following config files for our two *Experiments*:

```
# Contents of tests/resources/QC-Test-1.yml
```

```

QC Experiment 1:
  submit: True
  project: QC Project
  workflow: QC Workflow

  samples:
    - ESP001
    - ESP002

  protocols:
    - QC Quantification:
      data:
        ESP001:
          Type: DNA
          Ratio: 1.8
        ESP002:
          Type: RNA
          Ratio: 1.9

```

```
# Contents of tests/resources/QC-Test-2.yml
```

(continues on next page)

(continued from previous page)

```

QC Experiment 2:
submit: True
project: QC Project
workflow: QC Workflow

samples:
- ESP003
- ESP004

protocols:
- QC Quantification:
  data:
    ESP003:
      Type: DNA
      Ratio: 1.7
    ESP004:
      Type: RNA
      Ratio: 2.0

```

1.11.3 Step 3: Update Tests to Include *Experiments*

After creating these two configs, we can update our tests to run them after *Workflows* are created.

Update the original test file to look as follows:

```

# -*- coding: utf-8 -*-

# imports
import os
import unittest

from . import CONFIG, RESOURCES, CONTENT
from esp.testing import ModelLoaderMixin

# tests
class TestQCWorkflows(ModelLoaderMixin, unittest.TestCase):
    IMPORT = dict(
        Workflow=[
            os.path.join(CONTENT, 'workflows', 'QC-Workflow.yml'),
        ],
        Protocol=[
            os.path.join(CONTENT, 'protocols', 'QC-Quantification.yml'),
            os.path.join(CONTENT, 'protocols', 'QC-Report.yml'),
        ],
        Pipeline=[
            os.path.join(CONTENT, 'pipelines', 'QC-Report-Pipeline.yml'),
        ]
    )
    DATA = dict(
        Project=[
            {'name': 'QC Project'}

```

(continues on next page)

(continued from previous page)

```

    ],
    Experiment=[
        os.path.join(RESOURCES, 'QC-Test-1.yml'),
        os.path.join(RESOURCES, 'QC-Test-2.yml'),
    ]
)

```

Finally, **run the updated test** to get the following output:

```

~$ pytest tests/test_qc_workflow.py

===== test session starts _
↪=====
platform darwin -- Python 3.7.1, pytest-4.1.0, py-1.7.0, pluggy-0.8.0 -- /usr/local/opt/
↪python/bin/python3.7
cachedir: .pytest_cache
collected 1 item

tests/test_demos/test_quickstart.py::TestQuickstart::test__content
INFO:root:Clearing existing content from database.
INFO:root:Creating Task: Generate QC Report
INFO:root:Creating Pipeline: QC Report Pipeline
INFO:root:Creating PipelineReport: QC Report
INFO:root:Creating Protocol: QC Quantification
INFO:root:Creating Protocol: QC Report
INFO:root:Creating Workflow: QC Workflow
INFO:root:Creating Project: QC Project
INFO:root:Creating Sample: ESP001
INFO:root:Creating Sample: ESP002
INFO:root:Creating Experiment: QC Experiment 1
INFO:root:Submitting Experiment: QC Experiment 1
INFO:root:Creating SampleSheet: QC Experiment 1
INFO:root:Creating Sample: ESP003
INFO:root:Creating Sample: ESP004
INFO:root:Creating Experiment: QC Experiment 2
INFO:root:Submitting Experiment: QC Experiment 2
INFO:root:Creating SampleSheet: QC Experiment 2
Successfully imported config data.
PASSED

===== 1 passed in 8.54 seconds _
↪=====

```

From this test output, we can see that all of our testing data was defined correctly (along with our content).

1.12 Run Tests

1.12.1 Step 1: Run All Integration Tests

Now, putting it all together, **run all of the tests we've defined** in the L7|ESP SDK using the test make target:

```
~$ make test
```

This command will clear existing related content from the database, and run both the `test_status.py` and `test_qc_workflow.py` tests.

1.12.2 Step 2: Clear the Database

By default, artifacts created during tests are not removed after a test run so that developers can visit the application UI and see any resulting content. When tests are re-run, the test suite will automatically tear down content from a previous run and then proceed with executing tests. Usually, you won't need to clear the database between running tests, but if you want to run everything from a clean slate, you can use the commands below to clear the database outside of a test run. There are two ways to do this, both outlined in the *Administration* section of the documentation:

```
~$ # keep the application running and use
~$ # the python client to run tests
~$ make clear

~$ # completely wipe the database and reload
~$ # the application
~$ make reset
```

1.13 Seed Content

Finally, now that we've created and tested a *Workflow* that's ready to be deployed in our production L7|ESP instance, we can update our production configuration to import the *Workflow*. There are two components to this process:

1. Add configuration to a new or existing **seed** file that can be imported on install/update.
2. If necessary, add a reference to the **seed** file in our production configuration file, `roles/packages.yml`.

1.13.1 Step 1: Create Seed File to Load Content

As a first step, we'll need to create a **seed** file to reference our configuration. In the L7|ESP SDK, a **seed** file is a file used by the client to import configuration for specific models. In the file, for each item to import into the system, you specify configuration and a model that the configuration describes.

Create this example seed file for importing our *Workflow*, *Protocols*, and *Pipeline* defined above:

```
# Contents of: roles/workflows.yml

- model: Workflow
  data: $LAB7DATA/content/workflows/QC-Workflow.yml

- model: Protocol
```

(continues on next page)

(continued from previous page)

```

data: $LAB7DATA/content/protocols/QC-Quantification.yml
- model: Protocol
  data: $LAB7DATA/content/protocols/QC-Report.yml
- model: Pipeline
  data: $LAB7DATA/content/pipelines/QC-Report-Pipeline.yml

```

In the file above, \$LAB7DATA is an environment variable referencing the L7|ESP SDK root directory. It can be used both in development and production deployments to reference the same location.

1.13.2 Step 2: Update Deployment Config to Use Seed Data

Once we've created our **seed** file, **update the shared configuration file**, `roles/packages.yml`, to reference that seed file for the content L7|ESP will load into the system by default:

```

---
biobuilds:
  - bwa

seed:
  - '{{ sdk }}/roles/workflows.yml'

```

In the file above, `{{ sdk }}` is an Ansible variable referencing the L7|ESP SDK root directory. It can be used both in development and production deployments to reference the same location.

1.13.3 Step 3: Test Seeding Content Locally

Before deploying changes, we'll want to test that the seed files and changes to production configuration are properly imported by the software.

To do so, **use the import make target**:

```

~$ make import

```

Since the same commands are used to seed the content in development as in production, this practice is sufficient for ensuring that content will be imported the next time you update your production instance.

1.14 Summary

That's it! In summary, by going through this tutorial, you now know how to:

1. Create content in the L7|ESP SDK to meet various lab workflow and analysis needs.
2. Organize content in the L7|ESP SDK so it can be referenced during development.
3. Update L7|ESP SDK configuration to automatically load content on install.
4. Write tests for *Workflow* definition integrity and functionality.
5. Run tests to verify functionality.

1.15 Container Infrastructure

By adopting a container model, the L7|ESP platform aims to remain cloud-agnostic and simple to deploy.

Containers solve a number of problems ranging from immutable deployments to self-healing infrastructure.

This also means that there are many solutions for running and managing containers in production to choose from.

1.15.1 Kubernetes

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

The L7|ESP platform can be deploy to any Kubernetes cluster using the [Helm](#) package manager and the provided [Chart](#).



Amazon Elastic Kubernetes Service

Requirements

To install the L7|ESP Helm chart on Amazon Elastic Kubernetes Service (Amazon EKS), you must have:

- An AWS account to deploy to
- Access keys for the AWS account (see [Managing access keys for IAM users](#))

The `kubectl` and `aws` command line tools will be use respectively to programmatically access the Kubernetes cluster and the AWS account. To install the CLI tools, follow the link below and choose the correct operating sytem:

- [Kubernetes CLI](#)
- [AWS CLI](#)

Infrastructure as Code

CloudFormation Template

The latest CloudFormation template is as follows:

CloudFormation template

```

AWSTemplateFormatVersion: '2010-09-09'
Description: EKS cluster using a VPC with two public subnets

Parameters:
  EKSClusterName:
    Type: String
    Default: ""
    Description: The desired name of your AWS EKS Cluster.

  EKSVersion:
    Type: String
    Default: 1.21
    AllowedValues:
      - 1.16
      - 1.17
      - 1.18
      - 1.21
    Description: The desired version of your AWS EKS Cluster.

  EKSClusterName:
    Type: String
    Default: "NodeGroup01"
    Description: The desired name of your AWS EKS Node Group.

  NodeAutoScalingGroupDesiredCapacity:
    Type: Number
    Default: 2
    Description: Number of desired Worker Node.
    # MinValue: 1
    # MaxValue: 5

  NodeAutoScalingGroupMinSize:
    Type: Number
    Default: 1
    Description: Minimum size of Node Group ASG.

  NodeAutoScalingGroupMaxSize:
    Type: Number
    Default: 5
    Description: Maximum size of Node Group ASG. Set to at least 1 greater than
↳NodeAutoScalingGroupDesiredCapacity.

  EKSClusterName:
    Type: String
    Default: t2.large
    AllowedValues: [t2.nano, t2.micro, t2.small, t2.medium, t2.large, t2.xlarge, t2.
↳2xlarge,
      t3.nano, t3.micro, t3.small, t3.medium, t3.large, t3.xlarge, t3.2xlarge,
      m4.large, m4.xlarge, m4.2xlarge, m4.4xlarge, m4.10xlarge,
      m5.large, m5.xlarge, m5.2xlarge, m5.4xlarge,
      c5.large, c5.xlarge, c5.2xlarge, c5.4xlarge, c5.9xlarge,

```

(continues on next page)

(continued from previous page)

```
g3.8xlarge,r5.large, r5.xlarge, r5.2xlarge, r5.4xlarge, r3.12xlarge,
i3.xlarge, i3.2xlarge, i3.4xlarge, i3.8xlarge,
d2.xlarge, d2.2xlarge, d2.4xlarge, d2.8xlarge]
```

ConstraintDescription: Must be a valid EC2 instance type

Description: EC2 instance type for the node instances.

EKSIAmRoleName:

Type: String

Default: EKSClusterRole

Description: The name of the IAM role for the EKS service to assume.

EKSKeyPair:

Type: "AWS::EC2::KeyPair::KeyName"

Default: "devopskey"

Description: The name of Key Pair to establish connection with Worker Node.

VpcBlock:

Type: String

Default: 10.0.0.0/16

Description: The CIDR range for the VPC. This should be a valid private (RFC 1918) CIDR range.

↔CIDR range.

AllowedPattern: (\d{1,3})\.\(\d{1,3})\.\(\d{1,3})\.\(\d{1,3})/(\d{1,2})

ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

PublicSubnet01Block:

Type: String

Default: 10.0.0.0/24

Description: CidrBlock for public subnet 01 within the VPC.

AllowedPattern: (\d{1,3})\.\(\d{1,3})\.\(\d{1,3})\.\(\d{1,3})/(\d{1,2})

ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

PublicSubnet02Block:

Type: String

Default: 10.0.1.0/24

Description: CidrBlock for public subnet 02 within the VPC.

AllowedPattern: (\d{1,3})\.\(\d{1,3})\.\(\d{1,3})\.\(\d{1,3})/(\d{1,2})

ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

AvailabilityZonePublicSubnet01:

Type: CommaDelimitedList<AWS::EC2::AvailabilityZone::Name>

Default: us-east-1a

Description: Availability Zone for the Public Subnet 01.

AvailabilityZonePublicSubnet02:

Type: CommaDelimitedList<AWS::EC2::AvailabilityZone::Name>

Default: us-east-1b

Description: Availability Zone for the Public Subnet 02.

Metadata:

AWS::CloudFormation::Interface:

ParameterGroups:

-

Label:

(continues on next page)

(continued from previous page)

default: "Worker Network Configuration"**Parameters:**

- VpcBlock
- PublicSubnet01Block
- AvailabilityZonePublicSubnet01
- PublicSubnet02Block
- AvailabilityZonePublicSubnet02

-

Label:**default:** "EKS Cluster Information"**Parameters:**

- EKSClusterName
- EKSVersion
- EKSClusterName
- NodeAutoScalingGroupDesiredCapacity
- EKSClusterName
- EKSClusterName
- EKSClusterName
- NodeAutoScalingGroupMinSize
- NodeAutoScalingGroupMaxSize

Mappings:**ServicePrincipals:****aws-cn:****ec2:** ec2.amazonaws.com.cn**aws-us-gov:****ec2:** ec2.amazonaws.com**aws:****ec2:** ec2.amazonaws.com**Resources:****eksVPC:****Type:** AWS::EC2::VPC**Properties:****CidrBlock:** !Ref VpcBlock**EnableDnsSupport:** true**EnableDnsHostnames:** true**Tags:**

- **Key:** Name
Value: !Sub '\${AWS::StackName}-VPC'
- **Key:** Project
Value: aws-eks

eksInternetGateway:**Type:** AWS::EC2::InternetGateway**Properties:****Tags:**

- **Key:** Name
Value: !Sub '\${AWS::StackName}-InternetGateway'
- **Key:** Project
Value: aws-eks

(continues on next page)

(continued from previous page)

```

eksVPCGatewayAttachment:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    InternetGatewayId: !Ref eksInternetGateway
    VpcId: !Ref eksVPC

eksPublicRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref eksVPC
  Tags:
    - Key: Name
      Value: !Sub '${AWS::StackName}-RouteTable'
    - Key: Project
      Value: aws-eks

eksPublicRoute:
  DependsOn: eksVPCGatewayAttachment
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref eksPublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref eksInternetGateway

eksPublicSubnet01:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: !Ref AvailabilityZonePublicSubnet01
    MapPublicIpOnLaunch: true
    CidrBlock:
      Ref: PublicSubnet01Block
    VpcId:
      Ref: eksVPC
  Tags:
    - Key: Name
      Value: !Sub "${AWS::StackName}-PublicSubnet01"
    - Key: Project
      Value: aws-eks

eksPublicSubnet02:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: !Ref AvailabilityZonePublicSubnet02
    MapPublicIpOnLaunch: true
    CidrBlock:
      Ref: PublicSubnet02Block
    VpcId:
      Ref: eksVPC
  Tags:
    - Key: Name
      Value: !Sub "${AWS::StackName}-PublicSubnet02"

```

(continues on next page)

(continued from previous page)

```

    - Key: Project
      Value: aws-eks

eksPublicSubnet01RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref eksPublicSubnet01
    RouteTableId: !Ref eksPublicRouteTable

eksPublicSubnet02RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref eksPublicSubnet02
    RouteTableId: !Ref eksPublicRouteTable

eksSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Cluster communication with worker nodes
    VpcId: !Ref eksVPC
    Tags:
      - Key: Name
        Value: !Sub "${AWS::StackName}-SecurityGroup"
      - Key: Project
        Value: aws-eks

eksIAMRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - eks.amazonaws.com
          Action:
            - 'sts:AssumeRole'
    RoleName: !Ref EKSIAMRoleName
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/AmazonEKSClusterPolicy

eksCluster:
  Type: AWS::EKS::Cluster
  Properties:
    Name: !Ref EKSClusterName
    Version: !Ref EKSVersion
    RoleArn:
      "Fn::GetAtt": ["eksIAMRole", "Arn"]
    ResourcesVpcConfig:
      SecurityGroupIds:
        - !Ref eksSecurityGroup

```

(continues on next page)

(continued from previous page)

```

    SubnetIds:
      - !Ref eksPublicSubnet01
      - !Ref eksPublicSubnet02
    DependsOn: [eksIAMRole, eksPublicSubnet01, eksPublicSubnet02, eksSecurityGroup]

eksNodeInstanceRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - !FindInMap [ServicePrincipals, !Ref "AWS::Partition", ec2]
          Action:
            - "sts:AssumeRole"
    ManagedPolicyArns:
      - !Sub "arn:${AWS::Partition}:iam::aws:policy/AmazonEKSWorkerNodePolicy"
      - !Sub "arn:${AWS::Partition}:iam::aws:policy/AmazonEKS_CNI_Policy"
      - !Sub "arn:${AWS::Partition}:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
    Path: /

eksNodeGroup:
  Type: AWS::EKS::Nodegroup
  Properties:
    ClusterName: !Ref EKSClusterName
    NodeRole:
      "Fn::GetAtt": ["eksNodeInstanceRole", "Arn"]
    AmiType: AL2_x86_64
    InstanceTypes:
      - !Ref EKSWorkerNodeInstanceType
    NodegroupName: !Ref EKSNodeGroupName
    RemoteAccess:
      Ec2SshKey: !Ref EKSKeyPair
    ScalingConfig:
      MinSize: !Ref NodeAutoScalingGroupMinSize
      DesiredSize: !Ref NodeAutoScalingGroupDesiredCapacity
      MaxSize: !Ref NodeAutoScalingGroupMaxSize
    Labels:
      Project: aws-eks
    Subnets:
      - !Ref eksPublicSubnet01
      - !Ref eksPublicSubnet02
    DependsOn: [eksCluster, eksNodeInstanceRole]
Outputs:
  SubnetIds:
    Description: Subnets IDs in the eksVPC
    Value: !Join [ ",", [ !Ref eksPublicSubnet01, !Ref eksPublicSubnet02 ] ]
  SecurityGroups:
    Description: Security group for the cluster control plane communication with worker_
↪ nodes

```

(continues on next page)

(continued from previous page)

```

Value: !Join [ ",", [ !Ref eksSecurityGroup ] ]
VpcId:
Description: The eksVPC Id
Value: !Ref eksVPC

```

Parameters

- **EKSClusterName** (String) - The desired name of your AWS EKS Cluster.
- **EKSVersion** (String) - The desired version of your AWS EKS Cluster. (1.16, 1.17, 1.18, 1.21 - Default: 1.21)
- **EKSNodeGroupName** (String) - The desired name of your AWS EKS Node Group. (Default: NodeGroup01)
- **NodeAutoScalingGroupDesiredCapacity** (Number) - Number of desired worker nodes. (Default: 2)
- **NodeAutoScalingGroupMinSize** (Number) - Minimum size of Node Group ASG. (Default: 1)
- **NodeAutoScalingGroupMaxSize** (Number) - Maximum size of Node Group ASG. Set to at least 1 greater than NodeAutoScalingGroupDesiredCapacity. (Default: 5)
- **EKSWorkerNodeInstanceType** (String) - EC2 instance type for the node instances. (Default: t2.large)
- **EKSRoleName** (String) - Description: The name of the IAM role for the EKS service to assume. (Default: EKSClusterRole)
- **EKSKeyPair** (AWS::EC2::KeyPair::KeyName) - The name of Key Pair to establish connection with Worker Node. (Default: "devopskey")
- **VpcBlock** (String) - The CIDR range for the VPC. This should be a valid private (RFC 1918) CIDR range. (Default: 10.0.0.0/16)
- **PublicSubnet01Block** (String) - CidrBlock for public subnet 01 within the VPC. (Default: 10.0.0.0/24)
- **PublicSubnet02Block** (String) - CidrBlock for public subnet 02 within the VPC. (Default: 10.0.1.0/24)
- **AvailabilityZonePublicSubnet01** (CommaDelimitedList<AWS::EC2::AvailabilityZone::Name>) - Availability Zone for the Public Subnet 01. (Default: us-east-1a)
- **AvailabilityZonePublicSubnet02** (CommaDelimitedList<AWS::EC2::AvailabilityZone::Name>) - Availability Zone for the Public Subnet 02. (Default: us-east-1b)

Parameters File

Create a `parameters.json` file for use when deploying the CloudFormation template:

CloudFormation template

```

[
  {
    "ParameterKey": "NodeAutoScalingGroupDesiredCapacity",
    "ParameterValue": "2"
  },
  {
    "ParameterKey": "EKSNodeGroupName",

```

(continues on next page)

(continued from previous page)

```

    "ParameterValue": "NodeGroup01"
  },
  {
    "ParameterKey": "NodeAutoScalingGroupMinSize",
    "ParameterValue": "1"
  },
  {
    "ParameterKey": "NodeAutoScalingGroupMaxSize",
    "ParameterValue": "5"
  },
  {
    "ParameterKey": "EKSWorkerNodeInstanceType",
    "ParameterValue": "t2.large"
  },
  {
    "ParameterKey": "EKSClusterName",
    "ParameterValue": "l7eksdevops"
  },
  {
    "ParameterKey": "EKSKeyPair",
    "ParameterValue": "devopskey"
  }
]

```

Provisioning

Create a profile for use with AWS CLI:

```

~$ aws configure --profile <profile>
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
Default region name [us-east-1]:
Default output format [None]:

```

Switch to the correct AWS profile for current shell session:

```

~$ export AWS_PROFILE=<profile>

```

Deploy CloudFormation stack:

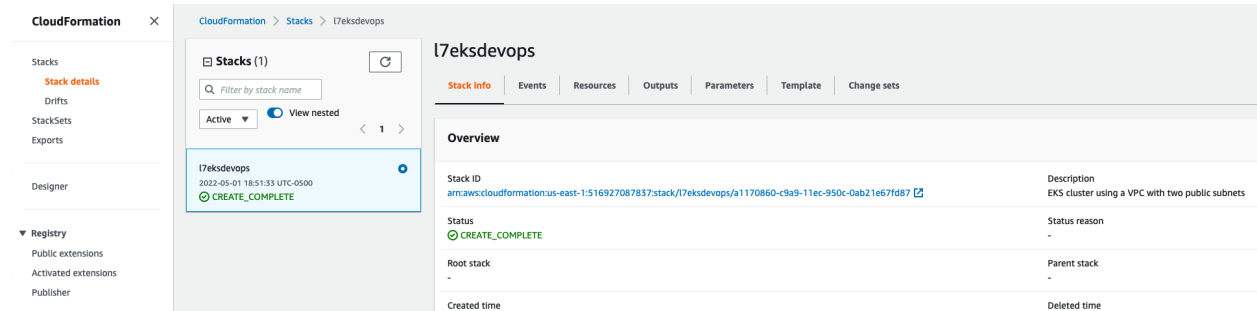
```

$ aws cloudformation deploy \
  --stack-name l7esp-example \
  --template-file eks.yml \
  --parameter-overrides file://eks.parameters.json \
  --capabilities CAPABILITY_NAMED_IAM

```

Validate Resource Creation

Log into the AWS Console and navigate to CloudFormation. Under “Stacks”, validate that the CloudFormation stack was created:



The screenshot shows the AWS CloudFormation console. On the left, the 'Stacks' section is active, displaying a list of stacks. The 'l7eksdevops' stack is highlighted, showing its status as 'CREATE_COMPLETE'. The main panel shows the 'Stack info' tab for 'l7eksdevops', with an 'Overview' section containing the following details:

Property	Value
Stack ID	arn:aws:cloudformation:us-east-1:1516927087837:stack/l7eksdevops/a1170860-c9a9-11ec-950c-0ab21e67fd87
Description	EKS cluster using a VPC with two public subnets
Status	CREATE_COMPLETE
Status reason	-
Root stack	-
Parent stack	-
Created time	-
Deleted time	-

Access the Kubernetes cluster

Use the AWS CLI to download the Kubernetes credentials into your local config:

```
~$ aws eks update-kubeconfig --name <stack_name>
Updated context arn:aws:eks:us-east-1:123456789012:cluster/<stack_name> in /home/<user>/.
kube/config
```

Check that Kubernetes nodes were successfully provisioned and are healthy:

```
~$ kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
ip-10-0-0-123.ec2.internal         Ready    agent    1h    v1.21.5-eks-9017834
ip-10-0-1-123.ec2.internal         Ready    agent    1h    v1.21.5-eks-9017834
```

You should see a list of nodes with a **Ready** status.

Installing L7|ESP Helm chart

To install L7|ESP on the new Kubernetes cluster, see the *Helm deployment guide*.



Azure Kubernetes Service

Prerequisites

The L7|ESP Helm chart will be installed on a managed cloud-native Kubernetes service, therefore it is imperative to have:

- Azure Subscription
- Azure Service Principal Account

CLI tools

The `kubectl` and `az` command line tools will be use respectively to programmatically access the Kubernetes cluster and the Azure subscription. To install the CLI tools, follow the link below and choose the correct operating system:

- [Kubernetes CLI](#)
- [Azure CLI](#)

ARM template

The latest ARM template is as follows:

Bicep

```

1  @description('The location of AKS resource.')
2  param azureLocation string = resourceGroup().location
3
4  @description('The name of the Managed Cluster resource.')
5  param clusterName string = 'l7esp-example'
6
7  @description('The number of nodes for the cluster. 1 Node is enough for Dev/Test and
8  ↳minimum 3 nodes, is recommended for Production')
9  @minValue(1)
10 @maxValue(100)
11 param clusterNodeCount int = 3
12
13 @description('The size of the Virtual Machine.')
14 param clusterNodeSize string = 'Standard_D4_v4'
15
16 @description('Disk size (in GiB) to provision for each of the agent pool nodes. This
17 ↳value ranges from 0 to 1023. Specifying 0 will apply the default disk size for that
18 ↳agentVMSize.')
19 @minValue(0)
20 @maxValue(1023)
21 param clusterNodeDiskGB int = 0
22
23 @description('Optional DNS prefix to use with hosted Kubernetes API server FQDN.')
24 param clusterDNSPrefix string = 'l7esp-example'
25
26 resource cluster 'Microsoft.ContainerService/managedClusters@2020-09-01' = {

```

(continues on next page)

(continued from previous page)

```

24 location: azureLocation
25 name: clusterName
26 tags: {
27   displayname: 'AKS Cluster'
28 }
29 identity: {
30   type: 'SystemAssigned'
31 }
32 properties: {
33   dnsPrefix: clusterDNSPrefix
34   agentPoolProfiles: [
35     {
36       name: 'agentpool'
37       osDiskSizeGB: clusterNodeDiskGB
38       count: clusterNodeCount
39       vmSize: clusterNodeSize
40       osType: 'Linux'
41       type: 'VirtualMachineScaleSets'
42       mode: 'System'
43     }
44   ]
45 }
46 }
47
48 output clusterFQDN string = cluster.properties.fqdn

```

JSON

```

1 {
2   "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.
↪json#",
3   "contentVersion": "1.0.0.0",
4   "metadata": {
5     "_generator": {
6       "name": "bicep",
7       "version": "0.6.18.56646",
8       "templateHash": "16874195123538177185"
9     }
10  },
11  "parameters": {
12    "azureLocation": {
13      "type": "string",
14      "defaultValue": "[resourceGroup().location]",
15      "metadata": {
16        "description": "The location of AKS resource."
17      }
18    },
19    "clusterName": {
20      "type": "string",
21      "defaultValue": "l7esp-example",

```

(continues on next page)

(continued from previous page)

```

22     "metadata": {
23       "description": "The name of the Managed Cluster resource."
24     }
25   },
26   "clusterNodeCount": {
27     "type": "int",
28     "defaultValue": 3,
29     "maxValue": 100,
30     "minValue": 1,
31     "metadata": {
32       "description": "The number of nodes for the cluster. 1 Node is enough for Dev/
↳ Test and minimum 3 nodes, is recommended for Production"
33     }
34   },
35   "clusterNodeSize": {
36     "type": "string",
37     "defaultValue": "Standard_D4_v4",
38     "metadata": {
39       "description": "The size of the Virtual Machine."
40     }
41   },
42   "clusterNodeDiskGB": {
43     "type": "int",
44     "defaultValue": 0,
45     "maxValue": 1023,
46     "minValue": 0,
47     "metadata": {
48       "description": "Disk size (in GiB) to provision for each of the agent pool nodes.
↳ This value ranges from 0 to 1023. Specifying 0 will apply the default disk size for
↳ that agentVMSize."
49     }
50   },
51   "clusterDNSPrefix": {
52     "type": "string",
53     "defaultValue": "l7esp-example",
54     "metadata": {
55       "description": "Optional DNS prefix to use with hosted Kubernetes API server.
↳ FQDN."
56     }
57   }
58 },
59 "resources": [
60   {
61     "type": "Microsoft.ContainerService/managedClusters",
62     "apiVersion": "2020-09-01",
63     "name": "[parameters('clusterName')]",
64     "location": "[parameters('azureLocation')]",
65     "tags": {
66       "displayname": "AKS Cluster"
67     },
68     "identity": {
69       "type": "SystemAssigned"

```

(continues on next page)

(continued from previous page)

```

70 },
71 "properties": {
72   "dnsPrefix": "[parameters('clusterDNSPrefix')]",
73   "agentPoolProfiles": [
74     {
75       "name": "agentpool",
76       "osDiskSizeGB": "[parameters('clusterNodeDiskGB')]",
77       "count": "[parameters('clusterNodeCount')]",
78       "vmSize": "[parameters('clusterNodeSize')]",
79       "osType": "Linux",
80       "type": "VirtualMachineScaleSets",
81       "mode": "System"
82     }
83   ]
84 }
85 ],
86 "outputs": {
87   "clusterFQDN": {
88     "type": "string",
89     "value": "[reference(resourceId('Microsoft.ContainerService/managedClusters',
90 ↪parameters('clusterName'))).fqdn]"
91   }
92 }
93 }

```

ARM template parameters

When deploying the ARM template, you can override parameters like so:

```

1 {
2   "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
↪deploymentParameters.json#",
3   "contentVersion": "1.0.0.0",
4   "parameters": {
5     "agentCount": {
6       "value": 4
7     },
8     "agentVMSize": {
9       "value": "standard_d11_v2"
10    }
11  }
12 }

```

- `azureLocation`: The location of AKS resource. Default value: same region/location as the resource group you are deploying ARM template into.
- `clusterDNSPrefix`: Optional DNS prefix to use with hosted Kubernetes API server FQDN. Default value: `aks-esp`
- `clusterName`: The name of the Managed Cluster resource. Default value: `aks101cluster-vmss`

- `clusterNodeCount`: The number of nodes for the cluster. One node is enough for Dev/Test and minimum 3 nodes is recommended for Production. Default: 3
- `clusterNodeDiskGB`: Disk size (in GiB) to provision for each of the agent pool nodes. This value ranges from 0 to 1023. Specifying 0 will apply the default disk size for that `agentVMSize`. Default value: 0
- `clusterNodeSize`: The size of the virtual machine. Default: `Standard_D4_v4`

Deploy ARM template

Login to Azure. For example, if you are using a service principle account that has Contributor role, you can login to Azure using the following command:

```
$ az login \
  --service-principal \
  --username <service-principal-id> \
  --password <service-principal-password> \
  --tenant <tenant-id>
```

Create a resource group to deploy into, if you haven't already:

```
$ az group create \
  --name <resource-group-name> \
  --location <resource-group-location>
```

Create ARM deployment in the resource group:

```
$ az group deployment create \
  --resource-group l7esp-example-rg \
  --name l7esp-example \
  --template-file ./aks.template.json \
  --parameters ./aks.parameters.json \
  --rollback-on-error \
  --verbose
```

Validate Resource Creation

In the Azure portal, navigate to the resource group and verify that all resources are listed:

The screenshot displays the Azure portal interface for a resource group named 'aks'. The left sidebar shows navigation options like Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Deployments, Security, and Policies. The main content area shows the 'Essentials' section with Subscription ID and Tags. Below that, the 'Resources' section is active, displaying a table with one resource: 'aks101cluster-vmss' of type 'Kubernetes service'. The 'Deployments' section at the top right shows a 'Succeeded' status for the deployment.

Access the Kubernetes cluster

Log into your Azure Subscription to access cluster:

```
$ az login
```

Use the Azure CLI to download the Kubernetes credentials into your local config:

```
$ az aks get-credentials \
  --name <cluster-name> \
  --resource-group <cluster-resource>
Merged "<my-cluster>" as current context in /home/<user>/.kube/config
```

Check that Kubernetes nodes were successfully provisioned and are healthy:

```
$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
aks-agentpool-21130059-vmss000000  Ready    agent    1h   v1.22.6
aks-agentpool-21130059-vmss000001  Ready    agent    1h   v1.22.6
aks-agentpool-21130059-vmss000002  Ready    agent    1h   v1.22.6
aks-agentpool-21130059-vmss000003  Ready    agent    1h   v1.22.6
```

You should see a list of nodes with a **Ready** status.

Installing L7|ESP Helm chart

To install L7|ESP on the new Kubernetes cluster, see the [Helm deployment guide](#).



Canonical MicroK8s

MicroK8s is the simplest production-grade conformant K8s. Lightweight and focused. Single command install on Linux, Windows and macOS.

Prerequisites

The L7|ESP Helm chart can be installed onto a multi-node MicroK8s cluster running on traditional VMs:

- A fleet of virtual machines (e.g. Ubuntu) to convert into MicroK8s nodes

Install Kubernetes

The installation steps for Ubuntu are, roughly, as follows:

```
$ sudo apt-get update
$ sudo apt-get upgrade -y
$ sudo snap install core
$ sudo snap install microk8s --classic --channel=1.20/stable
$ sudo microk8s status --wait-ready
$ sudo usermod -a -G microk8s l7esp
$ sudo chown -f -R l7esp ~/.kube
$ echo -e "l7esp\nl7esp" | passwd root
$ echo "alias kubect1='microk8s kubect1'" >> ~/.bash_rc
```

Note: Please refer to the official documentation for latest advice on how to install MicroK8s. The steps above are merely for demonstration purposes. [Install a local Kubernetes with MicroK8s](#)

Access the Kubernetes cluster

SSH into one of the cluster's master node and check that all nodes were provisioned and are in **Ready** state, with the following command:

```
$ microk8s kubect1 get nodes
NAME          STATUS  ROLES  AGE  VERSION
k8s-master   Ready  <none>  1h   v1.20.13-35+d877e7a8ac536e
k8s-node-1   Ready  <none>  1h   v1.20.13-35+d877e7a8ac536e
k8s-node-2   Ready  <none>  1h   v1.20.13-35+d877e7a8ac536e
```

Installing L7|ESP Helm chart

To install L7|ESP on the new Kubernetes cluster, see the *Helm deployment guide*.

Note: MicroK8s comes with its own packaged version of the Kubernetes CLI so remember to use the `microk8s kubect1` to ensure you are using the correct version when controlling the cluster.



Helm

Helm is a package manager for Kubernetes, and “charts” are the packages they install.

1. Confirm that you are connected to the correct Kubernetes cluster:

```
$ kubectl config use-context <cluster-name>
Switched to context "l7esp-example".
$ kubectl config current-context
l7esp-example
```

Note: If you do not have a *Kubernetes* cluster yet, please do that before proceeding.

2. Install Helm as documented on their website: https://helm.sh/docs/helm/helm_install/
3. Deploying the L7|ESP server to a Kubernetes cluster is performed using a Helm chart.

```
$ helm dependency update ./charts/l7esp
$ helm upgrade \
  --atomic \
  --create-namespace \
  --install \
  --namespace l7esp-example \
  --reset-values \
  --timeout 10m \
  --values ./values.yaml \
  l7esp-example \
  ./charts/l7esp
```

Note: You must provide an appropriate `values.yaml` file. All the configurable options are documented in the Helm chart `README.md`.

4. Validate L7|ESP server was installed by getting a list of all deployed Kubernetes resources:

```
$ kubectl get all --namespace <l7esp-namespace>
NAME                                READY   STATUS    RESTARTS   AGE
pod/l7esp-example-56595b4b8c-gmt98  1/1     Running   0           1h

NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)    ↵
↵-AGE
service/l7esp-example               ClusterIP      10.0.123.123    <none>         8002/TCP   ↵
↵-1h
```

(continues on next page)

(continued from previous page)

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/l7esp-example	1/1	1	0	1h
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/l7esp-example-56595b4b8c	1	1	0	1h

- If you want to see the logs of the L7|ESP server startup, you can do so by running:

```
$ kubectl logs --follow --namespace <l7esp-namespace> pod/l7esp-example-
↪ 56595b4b8c-gmt98
```

- Forward the L7|ESP server port to your local machine:

```
$ kubectl port-forward --namespace <l7esp-namespace> svc/l7esp-example-
↪ 8002:8002
```

- Verify that you can access the L7|ESP server's web UI in your browser by following this link:

localhost:8002

Setting up ingress

Ingress is a Kubernetes extension that allows you to expose services to the internet. The L7|ESP Helm chart does not install an Ingress controller, but you can install one yourself. A popular Ingress controller is the NGINX Ingress controller, which we will use in the following example:

- Deploy the Nginx ingress controller:

```
$ helm upgrade \
  --atomic \
  --create-namespace \
  --install \
  --namespace ingress-nginx \
  --reset-values \
  --values ./values/ingress-nginx.yaml \
  --wait \
  ingress-nginx \
  ingress-nginx/ingress-nginx
```

Note: The contents of `./values/ingress-nginx.yaml` are documented on the Nginx Ingress controller's website:

<https://kubernetes.github.io/ingress-nginx/>

- Update L7|ESP Helm chart values to apply any annotations required by the Ingress controller.

```
$ helm upgrade \
  --atomic \
  --create-namespace \
  --install \
  --namespace l7esp-example \
```

(continues on next page)

(continued from previous page)

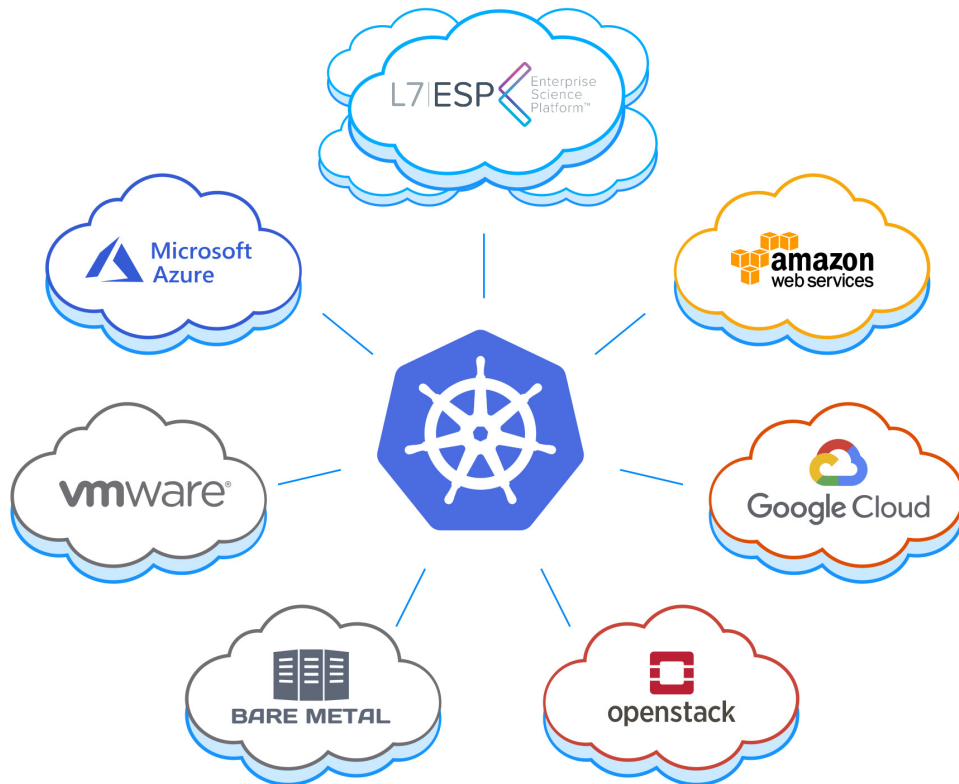
```
--reset-values \  
--values ./values.yaml \  
l7esp-example \  
./charts/l7esp
```

Note: See the `values.yaml` file in the L7|ESP Helm chart for more information.

3. Verify that the ingress controller is working by visiting the following URL:

<http://l7esp.example.com>

Note: If you are using a different domain name, you will need to update the ingress controller to use that domain name.



Amazon Elastic Kubernetes Service

Amazon Elastic Kubernetes Service (EKS) is a managed service and certified Kubernetes conformant to run Kubernetes

on AWS and on-premises.



Azure Kubernetes Service

Azure Kubernetes Service (AKS) offers serverless Kubernetes, an integrated continuous integration and continuous delivery (CI/CD) experience, and enterprise-grade security and governance.



Canonical MicroK8s

MicroK8s is the simplest production-grade conformant K8s. Lightweight and focused. Single command install on Linux, Windows and macOS.



Helm

Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

1.15.2 Docker



Azure Container Instances

Prerequisites

Azure Container Instances is a quick way to deploy your container into Docker host within Azure, without having to provision the underlying servers first (VM + OS).

- Azure Subscription
- Azure Service Principal Account

CLI tools

The az command line tool will be used to programmatically access your Azure subscription. To install the CLI tool, follow the link below and choose the correct operating system:

- [Azure CLI](#)

Obtain the latest ARM template

The latest ARM template is as follows:

Bicep

```
@description('Name (default: l7esp-example)')
param containerName string = 'l7esp-example'

@description('Azure location to deploy containers (default: Azure Resource Group,
↪location)')
param azureLocation string = resourceGroup().location

@description('L7|ESP software license (optional; JSON string)')
param appLicense string = ''

@description('L7|ESP admin password (required; initial password)')
param appPassword string = 'changeme'

@description('L7|ESP web UI port (default: 8002)')
param appPortHTTP int = 8002

@description('Container Image (e.g. acme/l7esp:3.0.0)')
param containerImage string = ''

@description('Container CPU cores (default: 2 vCPU)')
param containerCPU int = 2

@description('Container Memory (default: 6GB RAM)')
param containerMemory int = 6

@description('Container Restart Policy (default: Never)')
@allowed([
```

(continues on next page)

(continued from previous page)

```

    'Always'
    'Never'
    'OnFailure'
  ])
  param containerRestartPolicy string = 'Never'

  @description('Container Registry URL (default: docker.io)')
  param registryURL string = ''

  @description('Container Registry Username (e.g. Docker Hub user)')
  param registryUsername string = ''

  @description('Container Registry Password (e.g. Docker Hub app password)')
  param registryPassword string = ''

  resource container_instance 'Microsoft.ContainerInstance/containerGroups@2021-09-01' = {
    name: containerName
    location: azureLocation
    properties: {
      containers: [
        {
          name: containerName
          properties: {
            image: containerImage
            ports: [
              {
                port: appPortHTTP
                protocol: 'TCP'
              }
            ]
            environmentVariables: [
              {
                secureValue: appPassword
                name: 'L7ESP_PASSWORD'
              }
              {
                secureValue: appLicense
                name: 'L7ESP_LICENSE'
              }
            ]
            resources: {
              requests: {
                cpu: containerCPU
                memoryInGB: containerMemory
              }
            }
          }
        }
      ]
    }
  }
  osType: 'Linux'
  restartPolicy: containerRestartPolicy
  ipAddress: {

```

(continues on next page)

(continued from previous page)

```

    type: 'Public'
    ports: [
      {
        port: appPortHTTP
        protocol: 'TCP'
      }
    ]
  }
  imageRegistryCredentials: [
    {
      password: registryPassword
      server: registryURL
      username: registryUsername
    }
  ]
}
}

output containerIPv4Address string = container_instance.properties.ipAddress.ip

```

JSON

```

{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.
↪json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.6.18.56646",
      "templateHash": "17739852969827724894"
    }
  },
  "parameters": {
    "containerName": {
      "type": "string",
      "defaultValue": "l7esp-example",
      "metadata": {
        "description": "Name (default: l7esp-example)"
      }
    }
  },
  "azureLocation": {
    "type": "string",
    "defaultValue": "[resourceGroup().location]",
    "metadata": {
      "description": "Azure location to deploy containers (default: Azure Resource_
↪Group location)"
    }
  },
  "appLicense": {

```

(continues on next page)

(continued from previous page)

```

    "type": "string",
    "defaultValue": "",
    "metadata": {
      "description": "L7|ESP software license (optional; JSON string)"
    }
  },
  "appPassword": {
    "type": "string",
    "defaultValue": "changeme",
    "metadata": {
      "description": "L7|ESP admin password (required; initial password)"
    }
  },
  "appPortHTTP": {
    "type": "int",
    "defaultValue": 8002,
    "metadata": {
      "description": "L7|ESP web UI port (default: 8002)"
    }
  },
  "containerImage": {
    "type": "string",
    "defaultValue": "",
    "metadata": {
      "description": "Container Image (e.g. acme/l7esp:3.0.0)"
    }
  },
  "containerCPU": {
    "type": "int",
    "defaultValue": 2,
    "metadata": {
      "description": "Container CPU cores (default: 2 vCPU)"
    }
  },
  "containerMemory": {
    "type": "int",
    "defaultValue": 6,
    "metadata": {
      "description": "Container Memory (default: 6GB RAM)"
    }
  },
  "containerRestartPolicy": {
    "type": "string",
    "defaultValue": "Never",
    "allowedValues": [
      "Always",
      "Never",
      "OnFailure"
    ],
    "metadata": {
      "description": "Container Restart Policy (default: Never)"
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

},
"registryURL": {
  "type": "string",
  "defaultValue": "",
  "metadata": {
    "description": "Container Registry URL (default: docker.io)"
  }
},
"registryUsername": {
  "type": "string",
  "defaultValue": "",
  "metadata": {
    "description": "Container Registry Username (e.g. Docker Hub user)"
  }
},
"registryPassword": {
  "type": "string",
  "defaultValue": "",
  "metadata": {
    "description": "Container Registry Password (e.g. Docker Hub app password)"
  }
},
"resources": [
  {
    "type": "Microsoft.ContainerInstance/containerGroups",
    "apiVersion": "2021-09-01",
    "name": "[parameters('containerName')]",
    "location": "[parameters('azureLocation')]",
    "properties": {
      "containers": [
        {
          "name": "[parameters('containerName')]",
          "properties": {
            "image": "[parameters('containerImage')]",
            "ports": [
              {
                "port": "[parameters('appPortHTTP')]",
                "protocol": "TCP"
              }
            ],
            "environmentVariables": [
              {
                "secureValue": "[parameters('appPassword')]",
                "name": "L7ESP_PASSWORD"
              },
              {
                "secureValue": "[parameters('appLicense')]",
                "name": "L7ESP_LICENSE"
              }
            ]
          }
        }
      ],
      "resources": {

```

(continues on next page)

(continued from previous page)

```

        "requests": {
            "cpu": "[parameters('containerCPU')]",
            "memoryInGB": "[parameters('containerMemory')]"
        }
    },
    ],
    "osType": "Linux",
    "restartPolicy": "[parameters('containerRestartPolicy')]",
    "ipAddress": {
        "type": "Public",
        "ports": [
            {
                "port": "[parameters('appPortHTTP')]",
                "protocol": "TCP"
            }
        ]
    },
    },
    "imageRegistryCredentials": [
        {
            "password": "[parameters('registryPassword')]",
            "server": "[parameters('registryURL')]",
            "username": "[parameters('registryUsername')]"
        }
    ]
}
},
],
"outputs": {
    "containerIPv4Address": {
        "type": "string",
        "value": "[reference(resourceId('Microsoft.ContainerInstance/containerGroups', '...',
↳parameters('containerName'))).ipAddress.ip]"
    }
}
}
}

```

Deploy ARM template with desired parameters

When deploying the ARM template, you can override parameters like so:

```

{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.
↳json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "software_license": {
      "value": ""
    },
    "container_group_name": {

```

(continues on next page)

```
    "value": "espacicontainergroup"
  },
  "admin_password": {
    "value": ""
  },
  "image": {
    "value": "docker.io/l7esp/server:3.0.0-sdk.1"
  },
  "port": {
    "value": 8002
  },
  "cpuCores": {
    "value": 2
  },
  "server": {
    "value": "docker.io"
  },
  "credUsername": {
    "value": ""
  },
  "credPassword": {
    "value": ""
  },
  "memoryInGb": {
    "value": 6
  }
}
}
```

- `containerName` - Name (default: `l7esp-example`)
- `azureLocation` - Azure location to deploy containers (default: Azure Resource Group location)
- `appLicense` : L7|ESP software license (optional; JSON string)
- `appPassword` - L7|ESP admin password (required; initial password)
- `appPortHTTP` : L7|ESP web UI port (default: `8002`)
- `containerImage`: Container Image (e.g. `docker.io/acme/l7esp:3.0.0`)
- `containerCPU` : Container CPU cores (default: 2 vCPU)
- `containerMemory` : Container Memory (default: 6GB RAM)
- `containerRestartPolicy` - Container Restart Policy (default: `Never`)
- `registryURL` : Container Registry URL (default: `docker.io`)
- `registryUsername` : Container Registry Username (e.g. Docker Hub user)
- `registryPassword` : Container Registry Password (e.g. Docker Hub app password)

Run ARM template pipeline to create AKS

Login to Azure. For example, if you are using a service principle account that has Contributor role, you can login to Azure using the following command:

```
$ az login \
  --service-principal \
  --username <service-principal-id> \
  --password <service-principal-password> \
  --tenant <tenant-id>
```

Create a resource group to deploy into, if you haven't already:

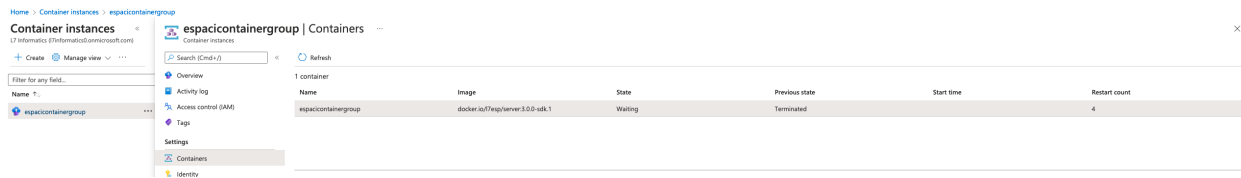
```
$ az group create \
  --name <resource-group-name> \
  --location <resource-group-location>
```

Create ARM deployment in the resource group:

```
$ az group deployment create \
  --resource-group l7esp-example-rg \
  --name l7esp-example \
  --template-file ./aci.template.json \
  --parameters ./aci.parameters.json \
  --rollback-on-error \
  --verbose
```

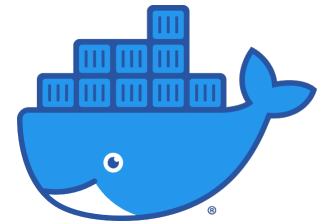
Validate Resource Creation

In the Azure portal, navigate to Container Instances and verify that the L7|ESP container instance was created and that it ends up in the **Running** state:



The screenshot shows the Azure portal interface for 'Container instances' in the resource group 'espaccontainergroup'. The table below displays the details of the container instances:

Name	Image	State	Previous state	Start time	Restart count
espaccontainergroup	docker.io/l7esp/venier:3.0.0-sdk.1	Waiting	Terminated		4



Docker Compose

Introduction

As of 3.0.0, L7|ESP is distributed as a container image. A container runtime, such as Docker, is the only requirement to run L7|ESP on any operating system. Cloud-based PaaS (platform as a service) solutions may also be used to run L7|ESP containers. In addition, container orchestration systems such as Kubernetes or Docker Swarm may be used to manage L7|ESP containers at scale. This particular guide assumes you are installing L7|ESP on a Linux server using the Docker container runtime. If this is not the case, please request documentation for your intended deployment approach.

Configure Linux

The simplest way to get started with L7|ESP is to use a Linux server with your operating system of choice. The first thing you will want to do is create a non-privileged Linux user that will be used to run the L7|ESP container and be the owner of any L7|ESP related files.

Optional L7|ESP User Account:

The following command should create a user called `l7esp` with home folder `/home/l7esp`:

```
$ adduser l7esp
```

The following command can be used to switch to this new user at any time:

```
$ sudo su - l7esp
```

Note: To get back to the root or previous, privileged user, simply type `exit` or press `Ctrl+D`.

Installing Docker

It is recommended that you follow the official installation documentation for your operating system: <https://docs.docker.com/engine/install/>

At the time of writing, Docker provides a convenience script at <https://get.docker.com/> that we will be using in this guide.

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

Follow any on-screen instructions to complete the Docker Engine installation. It is recommended to also review the post-installation instructions:

Optional, you can add the `l7esp` user to the `docker` group so it can run containers without requiring root or sudo privileges:

```
$ sudo usermod --append --groups docker l7esp
```

Note: For the Linux group change to take effect you will need to log out of your SSH session and back in again.

Installing Docker Compose

Docker Compose simplifies the running of containers by allowing you to specify all the container configuration in a YAML file, instead of running individual docker commands to create, configure and run containers.

It is recommended that you follow the official installation documentation for your operating system: <https://docs.docker.com/compose/install/>

At the time of writing, Docker Compose can be installed on Linux simply by downloading the executable:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.3.3/docker-
↪compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
```

Note: Replace the version v2.3.3 with the latest version available at the time of installation.

Installing L7|ESP Server

L7|ESP Deployment Bundle

Download and extract the archive provided to you by the L7 Informatics implementation team using roughly the following commands, but with the correct values where the dollar-signs are:

```
$ wget https://$download_link
$ tar xf $project-$version.tar.gz
```

L7|ESP Container Configuration

In the Deployment Bundle, there will be a Docker Compose configuration file that looks similar to the following:

```
$ cat docker-compose.yml
version: "3.9"
services:
  server:
    image: l7esp/server:3.0.0-rc.1
    environment:
      L7ESP_PASSWORD: "${L7ESP_PASSWORD?Set password in .env file}"
      L7ESP_LICENSE_FILE: "/opt/l7esp/data/project/conf/esp.license"
    ports:
      - published: "${L7ESP_PORT?Set port in .env file}"
        target: 8002
        protocol: tcp
        mode: host
    volumes:
      - type: volume
        source: data
        target: /opt/l7esp/data
      - type: bind
        source: .
        target: /opt/l7esp/data/project
```

(continues on next page)

(continued from previous page)

```
volumes:  
data:
```

Things to note in the above configuration:

- The `L7ESP_PASSWORD` and `L7ESP_PORT` environment variables are required and will be read from the `.env` file automatically. Make sure the `.env` file contains the initial administrator password you wish to use and that the port is set to 8002.
- The exact image name will be provided to you by L7 Informatics who will be able to set you up with access to pull the required image. This will require you to run the `docker login` command on the server. (Alternatively, if you have already downloaded an archive containing the image to use, you may use the `docker load` command.)
- You should replace `L7ESP_PASSWORD` environment variable value with the initial administrator password you wish to use. This will be the `admin@localhost` user unless you override the username with the `L7ESP_USER` environment variable.
- The published port 8002 will be the publicly exposed port that the L7|ESP web interface will be accessible on. It is recommended to set up a proxy server that provides TLS offloading before forwarding traffic to L7|ESP.
- The data volume is where L7|ESP will persist all data created at runtime and should be a part of your backup plan. You can determine the exact location of this volume by using the `docker inspect` command on the running container.
- The project bind-mount source path should point to the directory that contains your L7|ESP environment's project configuration and content. This will be provided to you by L7 Informatics either as a tarball that can be extracted or as a Git repository that can be cloned, depending on the use-case.

If you wish to make changes to the `docker-compose.yml` file, please communicate these changes with the L7 Informatics implementation team so they can incorporate these into the next archive you receive, such that the changes are permanent and well tested. Alternatively, you may create your own copy of the `docker-compose.yml` file in a different directory if you wish to maintain this yourself, provided you adjust the project bind-mount source path to point to the archive directory.

Starting L7|ESP

From the same directory as the Docker Compose configuration file (`docker-compose.yml`), run the following command to start the L7|ESP container in the background:

```
$ docker-compose up --detach
```

Next, run the following command to watch the logs as the container starts to ensure there are no errors:

```
$ docker-compose logs --follow
```

Once the log output has stopped, you can issue the following command to check if the L7|ESP web interface is healthy:

```
$ curl localhost:8002/health  
PASS
```

Finally, visit L7|ESP in your web browser of choice (Chrome is recommended):

```
http://SERVER_IP:8002
```

You should now be able to log in with the following credentials:

- Username: `admin@localhost` (or `L7ESP_USERNAME` value)

- Password: *Password12!* (or L7ESP_PASSWORD value)

Installing L7|ESP Content

So far, you should have the core L7|ESP server software running successfully. However, you will likely notice that the software is not licensed or otherwise configured correctly at this point.

The next step is to install your L7|ESP environment's project configuration and content, provided to you by the L7 Informatics implementation team.

First, get a shell into the L7|ESP container using the following command:

```
$ docker-compose exec server bash
```

Next, issue the the following commands to install the configuration and content:

```
$ make install
```

You will now see the a series of installation logs as the L7|ESP SDK, written mostly in Ansible, performs a series of task, including but not limited to:

Synchronize source files with the corresponding target directories in the container
Install your L7|ESP software license
Install any curated L7|Hub content bundles
Import custom configuration into the L7|ESP Config app
Import custom content into the L7|ESP Builders section
Hook any server extensions your configuration requires and reload services

Appendix

Migrating Data

If you are upgrading from an L7|ESP 2.x instance that was not containerized or otherwise need to restore the data volume from a backup, you can start the L7|ESP container while overriding the entrypoint command. This will disable the initialization process that usually runs, allowing you to perform the restoration.

Note: Replace *server_base* with installation directory where the previous L7|ESP server resides, which can be found by running the following command from the 2.x tarball directory:

```
$ grep server_base roles/prod.yml
```

- First, stop the existing L7|ESP 2.x server that you are planning on migration to containerized L7|ESP 3.x platform by running the following l7 stop command:

```
$ /server_base/Lab7_ESP/current/bin/l7 stop $(~/Lab7_ESP/current/bin/l7 status | grep -
↪v database | grep RUNNING | awk '{print $1}')
```

- Second, once the L7|ESP 2.x server has stopped, create a backup of the database and temporary copy of the Data directory that will be used during the restore process:

```
$ /server_base/Lab7_ESP/current/sys/bin/pg_dump \
  --host localhost \
  --port 1487 \
  --dbname lab7 \
  --clean \
```

(continues on next page)

(continued from previous page)

```

--if-exists \
--format=c > /tmp/backupesp_db_$(date).pgdump

$ rsync \
  --verbose \
  --archive \
  --exclude 'log/' \
  --exclude 'run/' \
  --exclude 'database/' \
  --exclude 'conf/' \
  /server_base/Lab7_ESP/Data/ \
  /tmp/backup

```

- Next, extract the L7|ESP 3.x deployment bundle tarball and change into the directory it unpacked (where docker-compose.yml resides) before running the rest of the commands:

```

$ wget https://$download_link
$ tar xf $project-$version.tar.gz
$ cd $project/

```

Note: You will need to log into Docker Hub with the appropriate credentials before proceeding using the docker login command. Please contact L7 Informatics for more information.

The following command will launch a one-off L7|ESP 3.x container with the configuration specified in docker-compose.yml (mainly, there will be a Docker volume created at /opt/l7esp/data) but with the following settings overridden by command-line flags:

- The entrypoint process will be overridden to the Bash shell, such that a) L7|ESP won't automatically initialize itself and run the server processes, and b) instead you will be granted a shell session where you can perform the restoration process
- The container will be run as the root user (UID 1) such that you don't run into Linux permission issues given that the files mounted in from the host may be owned by user(s) not present inside the container and you will later have to adjust these permissions.
- The container will be automatically removed when you leave the Bash shell such that it doesn't remain in the stopped state on the host after the restoration process, as we only need this container during the restore process and any files you move into the data volume will remain since the volume will be persisted.
- The backup taken earlier to /tmp/backup on the host machine will be bind-mounted to /tmp/restore in the container file system such that the files will be available for you to restore from once you have a shell session inside the container.

```

$ docker-compose run \
  --entrypoint bash \
  --user root \
  --rm \
  --volume /tmp/backup:/tmp/restore server

```

Copy the backup you mounted into this container to the location you just freed up:

```

$ rsync -avh /tmp/restore /opt/l7esp/data

```

Modify the Linux file ownership of the restored files to match the container's default user:

```
$ chown -Rfv l7esp:l7esp /opt/l7esp/data
```

The data has now been restored and you may now exit the one-off container. The container will be automatically removed but the volume with the restored data will remain:

```
$ exit
```

Now you can exit the one-off container and it will be automatically removed. The data volume will be kept, and will now contain the data you restored from the previous L7|ESP instance.

```
$ docker-compose up --detach && docker-compose logs --follow
```

Once the logs have finished initialization login to the container console to complete the database restore.

```
$ docker-compose exec server bash
```

Use the following commands to stop the current ESP instance and perform the DB restore

```
$ l7 stop $(l7 status | grep -v database | grep RUNNING | awk '{print $1}')

$ psql -h localhost -p 1487 -d postgres << EOF
DROP DATABASE lab7;
CREATE DATABASE lab7;
CREATE ROLE esp;
GRANT ALL PRIVILEGES ON DATABASE lab7 TO esp;
EOF

$ pg_restore \
  --host localhost \
  --port 1487 \
  --dbname lab7 \
  --clean \
  --if-exists \
  --no-owner \
  /opt/l7esp/data/backupesp_db_$(date).pgdump

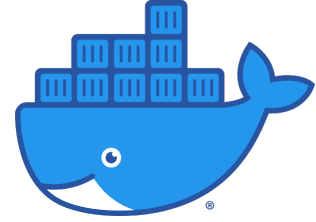
$ l7 init --yes

$ /usr/share/ansible/roles/l7esp_sdk/files/pg_esp_docker_migrate server_base

$ exit
```

Note: Replace *localhost* and *1487* with the appropriate hostname and PGSQL port when using a different ESP

See the Starting L7|ESP section above for commands to run to start up L7|ESP 3.0. The database you restored from 2.x will be automatically migrated to the latest 3.0 schema upon startup and if there are any errors you will see those in the container logs.



Docker Swarm

Prerequisites

To install L7|ESP onto a Docker Swarm cluster, you must have:

- At least 3 VMs with Docker Engine installed.

Note: As Docker Swarm is a distributed system, you must maintain a quorum of at least 3 nodes if you wish to be able to tolerate the failure of one of these nodes.

See [Add manager nodes for fault tolerance](#) in the official documentation for more information.

Creating the cluster

On the first node, run the following command to create the cluster:

```
$ docker swarm init
```

On the other nodes, run the following command to join the cluster:

```
$ docker swarm join-token manager
```

Note: It is recommended that the first two nodes are added as managers, per the fault tolerance recommendations above. Additional nodes can be added as workers also.

Compose file

Docker Swarm will take a Compose file as configuration so you should use the `docker-compose.prod.yml` file in root of your project, which will look something like this:

```
version: "3.2"
services:
  server:
    image: project_name/l7esp:3.0.0-sdk.1
    environment:
      L7ESP_PASSWORD: admin
    ports:
      - published: 8002
        target: 8002
```

(continues on next page)

(continued from previous page)

```

    protocol: tcp
    mode: host
  volumes:
    - type: volume
      source: data
      target: /opt/l7esp/data
volumes:
  data: {}

```

Deploy Compose stack

From the project folder, run the following command to deploy the stack.

```
$ docker stack deploy --compose-file=./docker-compose.prod.yml l7esp-example
```

Validate Resource Creation

From a Swarm manager node, run the following command. Remember that the password is 'vagrant'.

```

$ ssh manager.swarm.example.com
manager.swarm.example.com:~$ docker node ls
ID                                HOSTNAME          STATUS      AVAILABILITY  MANAGER STATUS
↪ENGINE VERSION
ri27kwen3bu9s8384cl45rq2w *      docker-manager    Ready      Active         Leader
↪20.10.16
i87w5vzik6qag5ia3e79d46c9       docker-worker1    Ready      Active
↪20.10.16

```

Validate L7|ESP Service

From a Swarm manager node, run the following command to validate that the L7|ESP container is running in the Swarm cluster:

```

$ ssh manager.swarm.example.com
manager.swarm.example.com:~$ docker node ps $(docker node ls -q)

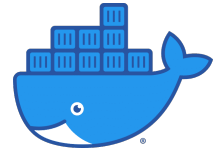
```

ID	NAME	IMAGE	NODE
↪DESIRED STATE	CURRENT STATE	ERROR	PORTS
e2fgtesxlr9	esp_server.1	l7esp/server:3.0.0-sdk.2	docker-worker1
↪Running	Running 1 minute ago		

Azure Container Instances

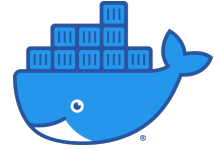


Azure Container Instances (ACI) allows you to run Docker containers in a managed, serverless cloud environment, without having to set up VMs, clusters, or orchestrators.



Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.



Docker Swarm

Docker in swarm mode is a container orchestration tool that allows you to manage multiple containers deployed across multiple host machines.

1.16 Content Installation

The sections below provide details on mechanisms for creating bundles and deploying content created with the L7|ESP SDK. After reading this documentation, you'll understand how production bundles of L7|ESP are built and how to install a production bundle in either an existing Linux server or inside of a container.

1.16.1 Deployment Bundle

For deploying the application in production, L7 provides an application bundle that contains specific versions of L7|ESP and related tools, along with all content that customers have defined in their L7|ESP SDK. These bundles must be generated on request by L7, and once created, can be used to install L7|ESP and custom content on a production system.

Deployment bundles are structured in a way that mirrors the L7|ESP SDK structure, since a lot of the commands for installing the bundle in production are the same as those that install the bundle in the development VM. Generally, the only differences between the L7|ESP SDK structure and the deployment bundle structure are a few boilerplate files (the `README.md`, `Makefile`, and the `docs/` folder).

1.16.2 Installation

Requirements

The deployment bundle must be installed on one of the following Linux-based operating systems:

- Ubuntu 20.04+
- RHEL/CentOS 7+

Aside from the operating system, the only additional requirement is *make*, which comes by default on most Linux-based operating systems.

Install

After extracting this package, you can use the Makefile to install L7|ESP and all custom content created for this deployment:

```
~$ tar -zxvf <package-name>.tar.gz
~$ cd <package-name>
~$ make install
```

This command will prompt for several configuration settings (hit <enter> to use defaults) and then proceed to install L7|ESP and import all custom content.

You can check the status of the installation at any time by running:

```
~$ make status
L7|ESP web server      : Available
L7|ESP system services : Available
L7|ESP database       : Available
```

Update

To update an existing installation with a new L7|ESP bundle or related software/content, you can use a similar process as the install process. However, instead of `make install`, you'll use `make update` to perform the update:

```
~$ tar -zxvf <package-name>.tar.gz
~$ cd <package-name>
~$ make update
```

This command will run through a lot of the same tasks as `make install`, but will preserve the database state and also perform any necessary database migrations for the update process.

1.16.3 Administration

Reload

If at any point you need to reload the application after install (i.e. if the application gets into a strange state or if you've rebooted the server), you can run:

```
~$ make reload
```

This command will take down the instance (if it's running) and reload the L7|ESP service (including all related web, logging, and pipeline services).

Import

If at any point you've deleted your database and need to re-import content defined in the customer bundle, you can run:

```
~$ make import
```

All of the content that you've defined as part of your production configuration will be loaded.

1.16.4 Database

Reset

If at any point you need to reset the application after install (i.e. completely wipe the application database), you can run:

```
~$ make reset
```

This command will take down the instance (if it's running), remove the database, re-run database migrations and setup, and reload the L7|ESP service (including all related web, logging, and pipeline services).

Note: Please be careful when resetting a production database. All data will be lost. If you set `db_archive: true` in your production playbook, the database will always be backed up in a folder called `archive` in the application install directory.

Archive

To create a backup of the application database (including all content and data created by the application), you can run:

```
~$ make archive
```

This command will create a `.sql` file (in a folder called `archive` inside of the application install directory) containing a database dump from the application.

1.16.5 Other Commands

Below are the list of available commands in the deployment bundle's Makefile:

```
~$ make help
info          list info about package
init         install prerequisites
install      provision production instance
update       update existing esp installation with ansible
reset        reset database with ansible tasks
archive      backup/archive esp database
stop         stop running esp instance
reload       reload or start application
import       import bundled content with ansible
status       check status of running esp instance
```

1.17 Hardware Recommendations

The most effective hardware configurations reduce application risk, but requirements will vary based on ESP instance activity. Consult with an L7 representative before committing to a configuration.

Note: Requirements are current as of October 2021 and are subject to change.

1.17.1 Baseline Configuration

Minimum recommended specifications to get up and running with a Single-Server setup. Suitable for test and development deployments.

- Server: 4-core, 16GB RAM, 500GB SSD storage
- Operating System: 64-bit Linux distro (e.g., RHEL 8, Ubuntu 20.04 LTS)

1.17.2 Performance Configuration

Increase performance with a two-server setup. Recommended minimum configuration for a production deployment.

- Application Server: 8-core, 32GB RAM, 500GB SSD storage
- Database server: 8-core, 32GB RAM, 500GB SSD storage
- Operating System: 64-bit Linux distro (e.g., RHEL 8, Ubuntu 20.04 LTS)
- Database: PostgreSQL 10.X,11.X

1.17.3 Optimal Performance Configuration

Minimize risk and increase performance with a customized multiple-server setup. (Does not include bioinformatics processing.)

Servers: The most reliable server configurations involve a combination of load balancers, shared storage, application servers, container orchestration software, and clustered database servers.

Minimum multi-server example:

- 2 x 8-core, 32GB RAM, 100GB disk application servers
- 2 x 8-core 32G RAM, 500GB disk database servers
- 1TB SSD class shared storage server
- Operating System: 64-bit Linux distro (e.g., RHEL 8, Ubuntu 20.04 LTS)
- Database: PostgreSQL 10.X,11.X

1.18 PostgreSQL Configuration

1.18.1 Introduction

The ESP application doesn't need any permissions outside of its own PostgreSQL database and it is recommended to create a single database for the application, as well as a dedicated role with matching name, and grant all permissions for that role to that database.

Currently, there isn't a role separation between "read-only", "read/write", "read/write/DDDL modification permissions" users at the application level. The same user is used with the *l7-init* (DDL migrations) as in normal application execution. These credentials are stored in the database config file */opt/l7esp/data/conf/database.json*.

Prerequisites 64-bit Linux VM with the following minimum specs:

- Debian or RHEL based OS
- 2vCPUs
- 8GB of RAM
- 30GB of available storage
- Timezone configured for UTC/GMT
- PostgreSQL version 10.x or 11.x

1.18.2 Instructions

ESP DB creation

Most installations of PostgreSQL done via a package manager include a user called "postgres". This user has full super admin privileges to the PostgreSQL instance installed on your system. Using the `su` command switch to the postgres user account then use the PostgreSQL interactive terminal (*psql*) to create the database, user role, and assign the permissions.

```
postgres=# CREATE DATABASE <esp_db_name>;
postgres=# CREATE USER <esp_db_user> WITH ENCRYPTED PASSWORD '<esp_db_password>';
postgres=# GRANT ALL PRIVILEGES ON DATABASE <esp_db_name> TO <esp_db_user>;
```

ESP DB configuration

The database configuration file at */opt/l7esp/data/conf/database.json* should take the following format when using an external hosted PostgreSQL solution such as AWS RDS:

```
{
  "host": "<postgres_server_url_or_ip>",
  "port": 5432,
  "user": "<esp_db_user>",
  "pass": "<esp_db_password>",
  "name": "<esp_db_name>",
  "start_service": false
}
```

Note: When using an external PGSQL server, it will either need the data migrated from the existing PGSQL server or the new PGSQL server will need to be initialized, to prepare it for use with ESP.

Preparing PostgreSQL for ESP

Once the ESP database configuration file has been configured to use an external DB and before starting ESP the PostgreSQL configuration file should be modified as follows for use with ESP. The location of the configuration file can be found in the PostgreSQL terminal by running the following query `SHOW config_file;`

PostgreSQL settings

- *max_connections*: L7 recommends that this setting should be a minimum of 6x the total number of ESP processes listed when running the `l7 status` command, L7 routinely sets this value to 100 and for systems configured with more web workers (the “http” processes) sets this value to 200.
- *shared_buffers*: This should be set to ~25% of the available system RAM*, but not more than 8GB. (*assumes you have a dedicated DB server OR enough RAM to handle the load of the DB + the app servers*).
- *temp_buffers*: Not less than 64MB. L7 routinely sets this to 128MB on dedicated DB servers in production.
- *work_mem*: Not less than 256MB. L7 routinely sets this to 1GB on dedicated DB servers in production.
- *maintenance_work_mem*: Not less than 128MB. L7 routinely sets this to 256MB on dedicated DB servers in production.
- *effective_cache_size*: On a dedicated DB server - 75% of available RAM. On other servers, 2x the shared_buffer size.

1.19 System Tuning

This section outlines some recommendations with respect to tuning L7|ESP application servers.

1.19.1 Operating System Tuning

As L7|ESP is cross-compiled and runs on many operating systems; we don't currently have any specific requirements for tuning at the operating system level (apart from hardware sizing recommendations which are in another document). Depending on the operating system, it's possible that some of the default values may not be suitable for a server environment. One such example of this pertains to the default per-user/application `ulimit` for open file descriptors, which can often be set to a very low value such as `1024` while a more suitable value for any server environment may be much higher (e.g. `24000`) depending on the amount of traffic the server is handling. Our recommendation is to increase these limits if you find you are bumping against them, however the exact values chosen may vary widely depending on scale.

1.19.2 Application Tuning

L7 recommends adjusting the following settings in a production L7|ESP installation, contingent on the server running the DB having adequate resources to do so.

Postgresql Configuration

The postgresql configuration (in *Lab7_ESP/Data/conf/postgresql.conf*) should be modified as follows:

1. **max_connections**: should be a minimum of 6x the total number of L7|ESP processes. For instance, if `l7 status` shows:

```
$ l7 status
l7-esp.broker RUNNING pid 95176, uptime 4 days, 4:44:44
l7-esp.concierge RUNNING pid 95184, uptime 4 days, 4:44:44
l7-esp.database RUNNING pid 95113, uptime 4 days, 4:44:46
l7-esp.executor RUNNING pid 95185, uptime 4 days, 4:44:44
l7-esp.haproxy RUNNING pid 95178, uptime 4 days, 4:44:44
l7-esp.http:l7-esp.http.0 RUNNING pid 95182, uptime 4 days, 4:44:44
l7-esp.http:l7-esp.http.1 RUNNING pid 95181, uptime 4 days, 4:44:44
l7-esp.http:l7-esp.http.2 RUNNING pid 95180, uptime 4 days, 4:44:44
l7-esp.logger RUNNING pid 95179, uptime 4 days, 4:44:44
l7-esp.notification RUNNING pid 95175, uptime 4 days, 4:44:44
l7-esp.pipeline RUNNING pid 95183, uptime 4 days, 4:44:44
l7-esp.scheduler RUNNING pid 95177, uptime 4 days, 4:44:44
```

Then you should set a minimum # of connections of 72. For 3-worker configurations, L7 routinely sets this value to 100, and for systems with more web workers (the “http” processes), L7 routinely sets the value to 200 for systems with more web workers.

2. **shared_buffers**: should be set to ~25% of the available system RAM*, but not more than 8GB. (*assumes you have a dedicated DB server OR enough RAM to handle the load of the DB + the app servers. See below for app server requirements).
3. **temp_buffers**: Not less than 64MB. L7 routinely sets this to 128MB on dedicated DB servers in production.
4. **work_mem**: Not less than 256MB. L7 routinely sets this to 1GB on dedicated DB servers in production.
5. **maintenance_work_mem**: Not less than 128MB. L7 routinely sets this to 256MB on dedicated DB servers in production.
6. **effective_cache_size**: On a dedicated DB server - 75% of available RAM. On other servers, 2x the **shared_buffer** size.

L7 also recommends indexing the following columns of the “resource_val” table (note: these indexes will be applied by default starting in L7|ESP 2.4):

1. The `bound_resource_id` column
2. The `step_instance_sample_id` column.

|esp| Worker Configuration

L7|ESP’s web workers are currently set, for most requests, to handle 1 request per worker*. Thus, the number of workers should be carefully considered when examining concurrent traffic load. (*Some routes such as adding/removing samples to/from LIMS worksheets allow concurrent requests to the same worker; the routes that support this will be expanding in L7|ESP 2.5).

At rest, an L7|ESP worker consumes ~250-500mb of RAM, depending on a number of implementation-specific factors. Under large workflow loads (large batch sizes and/or large workflows), the worker memory can spike to ~2GB of RAM. If you observe memory spikes in excess of 2-3GB per worker, please notify L7.

This means a baseline system with L7|ESP running 3 web workers will require ~6-8 GB of RAM for routine operations, excluding the DB needs, to properly service ~5 users.

Another consideration is automated processes (pipelines). In a 3 web worker configuration, all API requests from pipelines are sent to a single web worker and user requests may `_also_` be sent to this web worker. For production configurations, L7|ESP recommends a minimum of 4 workers. With 4 workers and a single “executor” thread for pipeline tasks, a maximum of one pipeline tasks will be executed at a time and all pipeline API requests will be sent to the fourth worker; all UI API requests will be routed to the first three workers. For a configuration supporting 10 users with a standalone DB server and an application server with 16GB of RAM, L7 recommends a minimum of 6 web workers as follows:

1. **Set the `num_workers` key in the `executor` block of L7|ESP configuration to 2** (this can be done via the `Config` app in the user interface)
2. **Set the `num_workers` key in the `web_server` block of the L7|ESP configuration to 6** (this can be done via the `Config` app in the user interface)
3. **Set the `numprocs` value of the `[program:l7-esp.http]` stanza of the `supervisord.conf` file to 6**
4. **Set the `-n` argument to 6 in the `command` key of the `[program:l7-esp.haproxy]` stanza.**

For systems where the DB server is co-located with the application server, the system should have a minimum of 32GB of RAM for production use in environments where large worksheets are anticipated. (For this document, a large worksheet is any worksheet processing > 96 samples through a workflow with a combined total # of columns >32 across all protocols. For instance, a server anticipating large loads should run with 32GB of RAM, an `effective_cache_size` (postgres.conf) or 16GB, `shared_buffers` of 8GB, and number of web workers = 3-6 and number of executors = 1-2 depending on anticipated concurrent user use.

HA Cluster Setup

L7|ESP application servers don’t store files in the database, instead writing these to disk and storing a reference to the file path in the database. This is the case for any pipeline scripts that are generated and executed, the `stdout/stderr` logs we persist for the runs of such pipelines, and files that are generated by these pipelines and/or user-uploaded to the system via the Data application or related API endpoints.

L7|ESP writes all such state to a directory in the server root named “data”. Depending on the installation, this may also contain logs generated by the running L7|ESP services and the PostgreSQL data directory if using the included local database. This works well for Dockerized use-cases where you may wish to make the filesystem read-only and capture all persistent state in a volume, though usually logging would be reconfigured to redirect to `stdout` in this case.

This can also be leveraged to create highly available architectures, where two or more L7|ESP application servers are placed behind a load balancer and are all accepting traffic, or you wish to have a hot standby that isn’t accepting traffic in the event that the primary application server experiences a failure. In such mode, you will want to mount this directory to a network storage location (such as AWS EFS) and careful consideration should be given to the `supervisord` configuration, particularly to where log files and PID files are written and, if written to a shared storage, if the names of such files will collide and/or cause multiple processes to write to them.

Your options here vary depending on whether you wish these services to log to disk, to syslog, to stdout (e.g. Docker) and, if logging to disk, if you wish to log to the shared storage or not (e.g. for backup purposes). The simplest recommendation in a HA setup is to adjust the `LAB7LOGDIR` and `LAB7VARRUN` environment variables to some directory outside of the Data directory, though it’s also possible to tailor logging in `supervisord.conf` to your exact requirements, such as including the `hostname` environment variable as part of the log filenames using the special `%(ENV_HOSTNAME)s` syntax.

In the event that the data volume should be mounted to a shared location but sharding occurs (e.g. the mount doesn't appear on one or more L7|ESP application servers for some reason and they begin writing these files to local disk), recovery should be as simple as merging the directories and files in these folders back together as all automatically generated file and directory names have either the database UUID they reference or a timestamp in their path or filename. Files that are not dynamically generated, such as custom Python scripts that pipelines may execute should always be the same on each application server as these are usually placed there at install time.

Database User and Grants

The L7|ESP application doesn't need any permissions outside of its own PostgreSQL database so it is recommended to create a single database schema for the application, as well as a dedicated role with matching name, and grant all permissions for that role to that schema.

Currently, we don't separate "read-only", "read/write", "read/write/DDL modification permissions" users at the application level. The same user is used in `l7 init` (DDL migrations) as in normal application execution. The credentials stored in the database config file `$LAB7DATA/conf/database.json` are used in both instances.

However, it is possible to switch these credentials with a more privileged account, run `l7 init` to perform any database migrations and then switch back to the less privileged account credentials for day-to-day application execution. Note that database migrations will only occur if you are upgrading the underlying L7|ESP version (e.g. L7|ESP 2.3.3 to L7|ESP 2.4) but won't take place during normal upgrades that only seed new content.

The database configuration file at `$LAB7DATA/conf/database.json` should take the following format when using an external hosted PostgreSQL solution such as AWS RDS:

```
{
  "host": "postgres",
  "port": 5432,
  "user": "l7esp",
  "pass": "password",
  "name": "l7esp",
  "start_service": false
}
```

Regarding encryption in transit, the PostgreSQL driver will automatically negotiate an appropriate connection based on the `SSL mode` defined on the database server.

Unix Application User

L7|ESP should be installed in Linux user-space and by default will only listen on user/registered ports, rather than system/well-known ports below 1024 that require root privileges.

The installation requires some basic tools, such as `make`, `wget`, `bzip2`, and `rsync`. This is enough for the installer to use Miniconda to bootstrap a Python environment where Ansible will be installed and automate the complete L7|ESP installation.

As root, you should install the basic requirements, and create a non-privileged Linux user/group for the application, and switch to that user:

```
root@l7espapp:~# yum install make wget bzip2 rsync
root@l7espapp:~# groupadd l7esp
root@l7espapp:~# useradd --groups l7esp l7esp
root@l7espapp:~# su - l7esp
```

As the L7|ESP user, you may fetch the deployment bundle, extract it and perform the installation:


```

17esp@17espapp:~$ wget "${ESP_DEPLOYMENT_BUNDLE_URL}"
17esp@17espapp:~$ tar xf "${ESP_DEPLOYMENT_BUNDLE_FILENAME}"
17esp@17espapp:~$ cd "${CUSTOMER_NAME}"
17esp@17espapp:~/${CUSTOMER_NAME}$ make install

```

To aide in system administration tasks, you may additionally wish to add the following lines to this Linux user's Bash profile so that common utilities will be made available via \$PATH the path environment variable, as well as some other useful environment variables:

```

17esp@17espapp:~$ cat ~/.profile
source /data/ESP/Lab7_ESP/current/bin/env.sh
PATH="/data/ESP/client/bin:$PATH"

17esp@17espapp:~$ which l7
/data/ESP/Lab7_ESP/current/bin/l7

17esp@17espapp:~$ which esp
/data/ESP/client/bin/esp

17esp@17espapp:~$ env | grep LAB7
LAB7LOGDIR=/data/ESP/Lab7_ESP/Data/log

17esp@17espapp:~$ tail -n 0 -f $LAB7LOGDIR/\*
==> /data/ESP/Lab7_ESP/Data/log/l7-esp.http.0.access.log <==

```

If for some reason, you need to migrate an existing L7|ESP instance to a different Linux account on the same machine (or any other machine for that matter), the steps would be as follows:

1. Stop the running L7|ESP instance with the `l7 stop` command.
2. If the L7|ESP server is installed in a user's home directory, move the entire server root directory to the target user's home directory.
3. Change file ownership on the entirety server root directory to the new Linux user/group (e.g. `chown -R 17esp:17esp /data/ESP`).
4. Start the L7|ESP instance back up again with the `l7 start` command.

Application Startup on Boot

The deployment bundle has the ability to install L7|ESP as a systemd service, however this will require the application user to be able to escalate to sudo privileges and that the Ansible variable `service: True` is set in the `roles/container.yml` file in your L7|ESP deployment bundle. You may also find the template used to create the service in the file `roles/esp/templates/service` and the related Ansible tasks can be found in the `roles/esp/tasks/run.yml` file. If you wish to make changes to any of this, please let us know so that we may include and test your desired defaults in future L7|ESP deployment bundle revision you receive from us.

L7|ESP Revision History

Simply put, all previous versions of workflows, protocols, and entities (samples) are stored in the database. In terms of old/new data values for a given field, L7|ESP captures the full history of changes in the `resource_action` table.

The `resource_action` database table provides an audit log for each resource in the system, which can typically be viewed under the History tab in the UI for a given resource.

When it comes to configuration items, such as workflow and protocol versions: a given version of a protocol (e.g.) is immutable. Each time you save a protocol or workflow, you are actually creating a new version of the protocol under the hood. The old version remains unmodified.

Currently, experiments use the most recent version of a workflow, in addition to the most recent version of the protocols nested in this workflow. Once submitted, the samples included in the experiment are “locked” into these versions - subsequent changes to the workflow or protocol definitions will not impact any “in-flight” samples.

1.20 Backup and Disaster Recovery

1.20.1 Introduction

This document outlines strategies for backup, recovery and DR, as it pertains to the ESP server application. These suggestions can be implemented to achieve highly available (HA) infrastructure as well as help inform decisions when performing risk analysis related to your own internal business continuity (BC) guidelines and policies.

1.20.2 Risk Analysis

Determining the correct level of redundancy, HA, and backups should be defined by internal IT/DevOps/Business policies and processes to determine RTO (Recovery Time Objective) and RPO (Recovery Point Objective) values to meet any organizational defined SLO (Service Level Objective). One possible decision factor is the cost of implementing redundancy or HA versus lost revenue, reputation, and internal staff time associated with downtime.

1.20.3 Backup

Recommendations and examples

For atomic/PITR (*Point in Time Recovery*) backups, we recommend always backing up the database first, followed by the shared data volume (`/opt/l7esp/data`). This will ensure that all files referenced by the database backup are included in the whole backup.

At minimum L7 recommends whether performing a “hot” or “cold” backup:

- Keep copies of each deployment bundle that you deploy, since these contain information about the version of the software, as well as the configuration that was applied.
- Using `pg_dump --format="c"` for PostgreSQL database backups so you may be able to restore the backups with the `pg_restore` command. If you prefer to create backups in a different format with the `pg_dump` command, note that you will likely have to pipe the backup file into the `psql` PostgreSQL command-line utility to perform a restore.
- If the data volume exists outside of the default location (e.g. NFS/EFS mount) this should be backed up as well.
- Backup the deployed installation tarball to be able to reinstall if required.

One example of a backup strategy is to first backup the database to the shared data volume, then perform a backup of the data volume which will always result in atomic backups.

An option for AWS provisioned environments is the use of a managed service such as AWS Backup, which offers the following features:

- Centralized backup management
- Policy-based backup solution
- Tag-based backup policies
- Automated backup scheduling
- Automated retention management
- Backup activity monitoring
- Lifecycle management policies
- Incremental backups
- Backup data encryption
- Backup access policies
- Amazon EC2 instance backups
- Item-level recovery for Amazon EFS
- Cross-region backup
- Cross-account backup

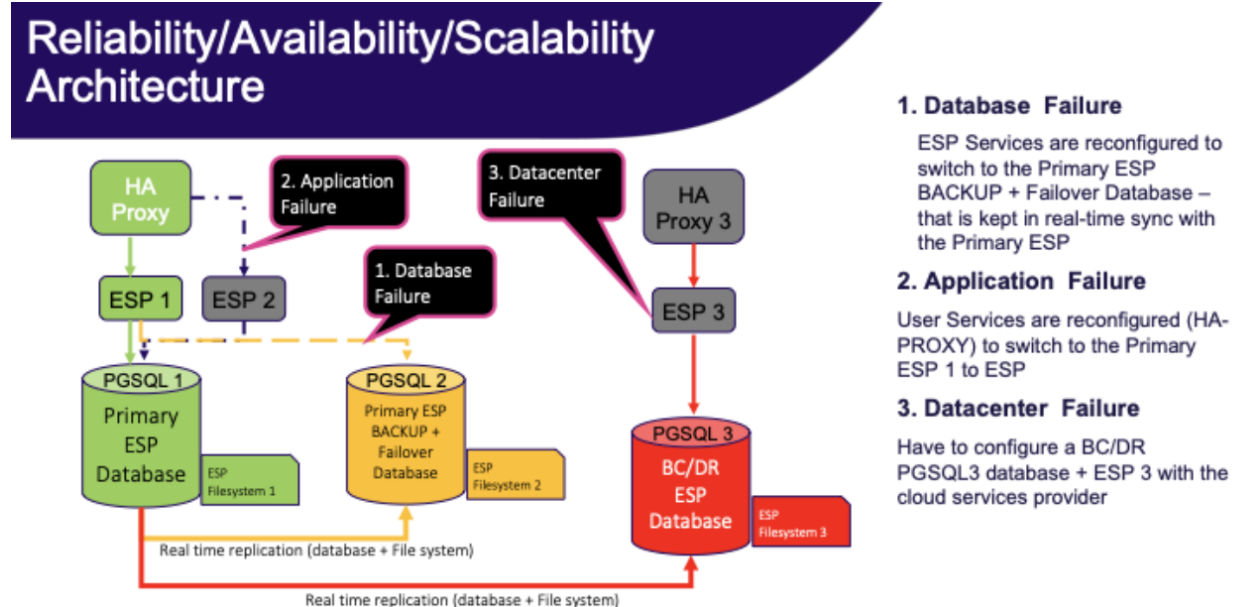
One thing to keep in mind when utilizing a managed service to perform a “hot backup” is that any RDS backup occurs before any EBS/EFS/NFS volumes to provide a valid restore point.

It is also highly recommended to regularly audit and test your backup/restore strategy to ensure it can be performed successfully as well as complies with any organizational policies and regulatory controls.

Common disaster-recovery instance requirements

- The disaster-recovery installation must have the same product version and patch level as the production installation.
- If any configuration or file changes (such as applying patches) are made to the production instance, the same changes must be repeated on the disaster-recovery instance.
- As the production system is used, all data changes must be replicated to the disaster-recovery instance. These changes can be database changes or file system changes, depending on the product in use.
- Replicating data changes imposes additional demands on the resources in the production system. To keep these demands to a minimum, the replication schedule should be carefully considered. If continuous replication is needed, the production system must be given additional resources (CPU and memory) to reduce the performance impact.

1.20.4 Disaster Recovery



Definitions

- **Hot standby** is a server that will automatically failover if the primary server fails.
- **Warm standby** is a server that will not automatically failover and that may not have all the latest transactions.
- **Cold standby** is a spare machine that needs to be turned on, backup restored (or even full staging of the machine).

Scenarios

Application failure

In the event of an ESP application server failure:

- **Hot standby:** Two ESP application servers behind a load balancer.

Note: User-uploaded files and pipeline scripts/log files that are referenced by the database are written as physical files to disk and are not stored inside the database as blobs for a number of reasons, such as performance. You should take care to mount these directories to a networked storage solution, such as an AWS EFS filesystem.

- **Warm standby:** Alternatively, you may cut traffic to another ESP application server of the same version, and sync the files in the ESP data volume using an out-of-band process, such as scheduled rsync. Using physical disks vs networked storage will increase file system performance and may reduce cost, but the file synchronization process will likely be eventually-consistent in nature.
- **Cold standby:** For this scenario, in the event of a failure, you can quickly bootstrap a new ESP application server either by installing a tarball on a fresh Linux server. Alternatively, if you create an image (e.g. AWS AMI) for an installed server, you can create a new VM from this image or even automate the process by using an AWS ASG with the correct policy and health checks.

Database failure

In the event of a PostgreSQL database server failure:

- **Hot standby:** The main use case for a hot standby is load balancing. You would use this to reduce to load on the database master server by delegating requests to one of more standby servers. To configure this, you must increase `wal_level` to `hot_standby` on the master database server and set `hot_standby` to on at the standby database servers. At a high level, most “clustered” PostgreSQL configurations can be considered hot standby, such as using more modern streaming replication modes.
- **Warm standby:** You can transfer a PITR backup to a standby database server and set it to always run an endless recovery process using WAL logs from the master database server. In this configuration, the standby database server is not accepting queries and sharing the load, but can be made available in the event of a failure. To configure this, you must specify `wal_level=replica`; `archive_mode=on` on the master database server and set `standby_mode` to on at the standby database server(s). When using a hosted service such as AWS RDS, you can simply enable the “Multi-AZ deployment” option when provisioning your database server.
- **Cold standby:** In this failure scenario, you would be configuring a new database server and restoring from backup. In AWS RDS, this would be equivalent to restoring a database instance from a snapshot.

Datacenter failure

In the event of a catastrophic datacenter (or regional, in cloud terms) failure:

- **Hot standby:** To achieve this scenario, you must have duplicate infrastructure running in another datacenter or region, with database and file synchronization between these sites. There is a cost vs risk tradeoff to be made as this can cost up to double the price, whereas the likelihood of this event may not mandate automated failover at this level.
- **Warm standby:** This is the same as a hot standby from a cost perspective as you would have duplicate infrastructure running, however you are failing over to the standby site by manually altering DNS records in the event of a failure.
- **Cold standby:** In this failure scenario, you would have a spare machine or the ability to provision one, in another network. The ESP application would need to be installed and backups restored to make it functional, and DNS updated to direct traffic here afterwards. In AWS, you could spin up a copy of the ESP infrastructure in another region using existing IaC in the unlikely event that this scenario ever occurs.

1.21 Questions/Feedback

For any questions about content development, testing, configuration, or anything else about the L7|ESP SDK, please contact L7 Informatics at support@l7informatics.com.