Welcome

EIC EDGE – Week 2

Software Development Practice

DevelopIntelligence
A PLURALSIGHT COMPANY

Why study these subjects?

Being able to effectively apply your software development methodology in practice can be the difference between success and failure

- Making judicious use of automated unit tests helps you practice “shift left” and high quality
- Knowing how to troubleshoot issues with code as you develop helps you complete tasks and find and fix bugs
- Understanding key deployment technologies like Docker and Kubernetes helps you deliver software faster
- Understanding principles of good code review helps you contribute to overall team success

My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Use an on-screen timer

Objectives

At the end of this course, you will be able to:

- Write automated unit tests in Python, measure code coverage as part of a TDD-strategy, and troubleshoot issues encountered when building Python apps
- Use Docker to containerize workloads, enabling consistent delivery of high-quality software regardless of deployment target
- Contribute to team productivity by effectively reviewing and providing feedback on code changes

Agenda

- **Day One**
- Python Refresh, Part 1

- **Day Two**
- Python Refresh, Part 2
- Unit Testing/Debugging in Python, Part 1

- **Day Three**
- Unit Testing/Debugging in Python, Part 2

Agenda

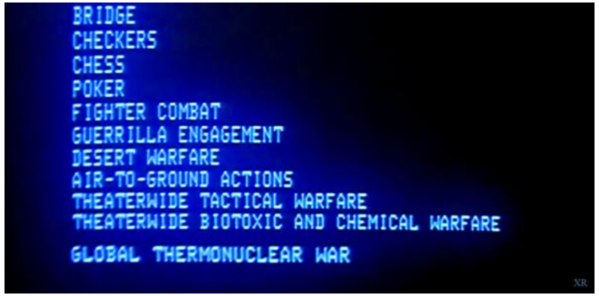
- **Day Four**
- Docker & Kubernetes, Part 1

- **Day Five**
- Docker & Kubernetes, Part 2

How we’re going to work together

- Lab work will take place in individual workspaces – isolated, browser-based virtual machines
- Classroom labs may include individual work as well as group discussion and work
- I welcome being interrupted – if you need more info, or clarification, or anything else, just break in and ask. I am here to help you.

Would you like to play a game?


BRIDGE
CHECKERS
CHESS
POKER
FIGHTER COMBAT
GUERRILLA ENGAGEMENT
DESERT WARFARE
AIR-TO-GROUND ACTIONS
THEATERWIDE TACTICAL WARFARE
THEATERWIDE BIOTOXIC AND CHEMICAL WARFARE

GLOBAL THERMONUCLEAR WAR

- To make it interesting, let's add a competition factor to our time together ☺
- For questions posed to the larger group, 1 point will be awarded for each answer provided by a participant
- 2 points will be awarded for acting as a spokesperson for a breakout room during one of our group discussions
- There will be a rubric provided for the capstone (targeted for week 4) which will define a points breakdown for the implemented solution

Python Refresh

Data Types, Operators, and Variables

https://github.com/KernelGamut32/intermediate-python/blob/main/01%20Data%20Types%2C%20Operators%2C%20and%20Variables.ipynb

Conditionals

https://github.com/KernelGamut32/intermediate-python/blob/main/02%20Conditionals.ipynb

Functions

https://github.com/KernelGamut32/intermediate-python/blob/main/03%20Functions.ipynb

Looping

https://github.com/KernelGamut32/intermediate-python/blob/main/04%20Looping.ipynb

Additional Datatypes

https://github.com/KernelGamut32/intermediate-python/blob/main/05%20Additional%20Datatypes.ipynb

Files and Regular Expressions

https://github.com/KernelGamut32/intermediate-python/blob/main/06%20Files%20and%20Regular%20Expressions.ipynb

Modules

https://github.com/KernelGamut32/intermediate-python/blob/main/07%20Modules.ipynb

Object-Oriented Programming

https://github.com/KernelGamut32/intermediate-python/blob/main/08%20Object-Oriented%20Programming.ipynb

Unit Testing/Debugging in Python

Common Components of a Unit Testing Framework

- Base class defined in the framework from which test classes derive
- Fixtures that run before and after blocks of test code
- Assertions – the automated evaluation executed against our code to verify
- Test suite that collects a group of related tests to be executed together
- Runner built-in to the framework that executes the tests
- Reporting mechanism that can provide results in different formats

Arrange, Act, Assert (AAA)

Pattern used in unit testing to help us organize our tests, improve readability, and ensure good practice when building tests.

Arrange

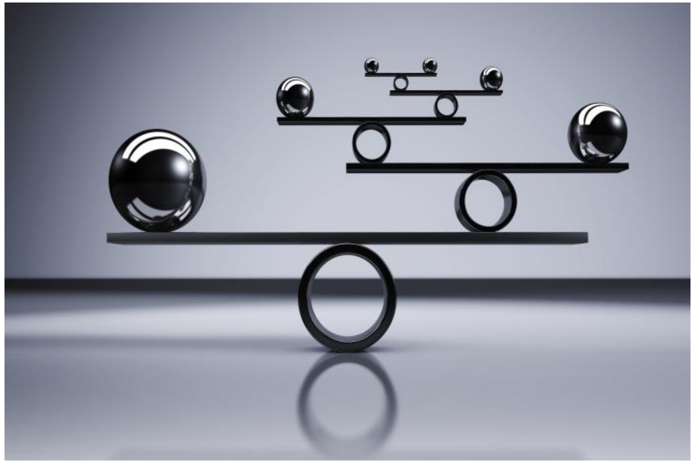
- Execute any required set up in preparation for running one or more tests
- Can include creation of objects that will be needed during test execution
- Might also include set up of mocks (to ensure test isolation) or creation of test data (if directly testing data access methods)

Act

- Exercising the code under test
- Executing any methods or actions on the class structures you want to verify
- Representative of how code might be called in production
- Should not include an extensive amount of code – otherwise, could be a “smell” that you’re testing too much in a given test case

Assert

- Does the result match my expectation for how the code under test should function?
- Determines pass/fail for the test
- Often tests will have only a single assert
- Can include multiple asserts, but the asserts should all relate cohesively to the same function/test case/object



Writing Unit Tests in unittest

unittest Framework

- Included in Python standard library
- Provides a framework for building and running unit tests against Python code (functions, classes, etc.)
- A Python file houses code that will be used to automate the testing

unittest Framework

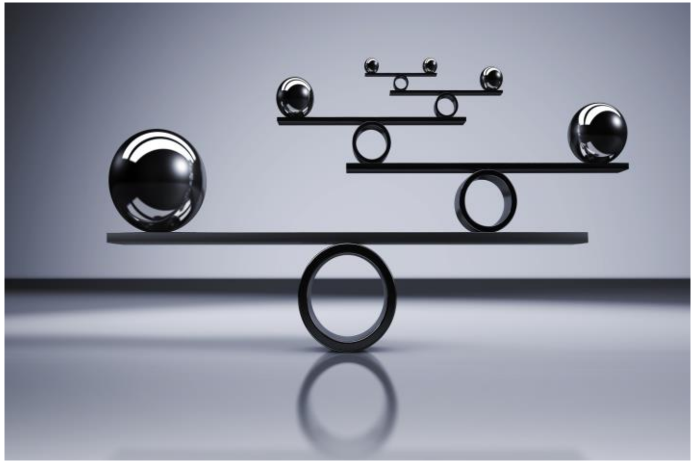
- Test code imports unittest and module under test
- We write Python methods to exercise our business logic and verify expected outputs

unittest Framework - Keys

- Our test class derives from *unittest.TestCase*
- Several versions of “assert” method provide robust validation
- Usually, test code and code under test kept in separate files (and potentially in separate folders)
- Test fixtures promote reuse and help keep our code clean
- unittest offers multiple options for specifying which of a set of tests you want to execute

unittest Framework

- unittest includes multiple “assert” methods for checking different kinds of results (expected vs. actual)
- Can exercise tests through unittest framework
- Results will indicate success (or failure)



unittest Framework

Assert methods include (among others):

- assertEqual/assertNotEqual
- assertTrue/assertFalse
- assertIn/assertNotIn

LAB:

unittest Assert Methods

Review the documentation on the different types of assert methods available with unittest (at https://docs.python.org/3/library/unittest.html#assert-methods).

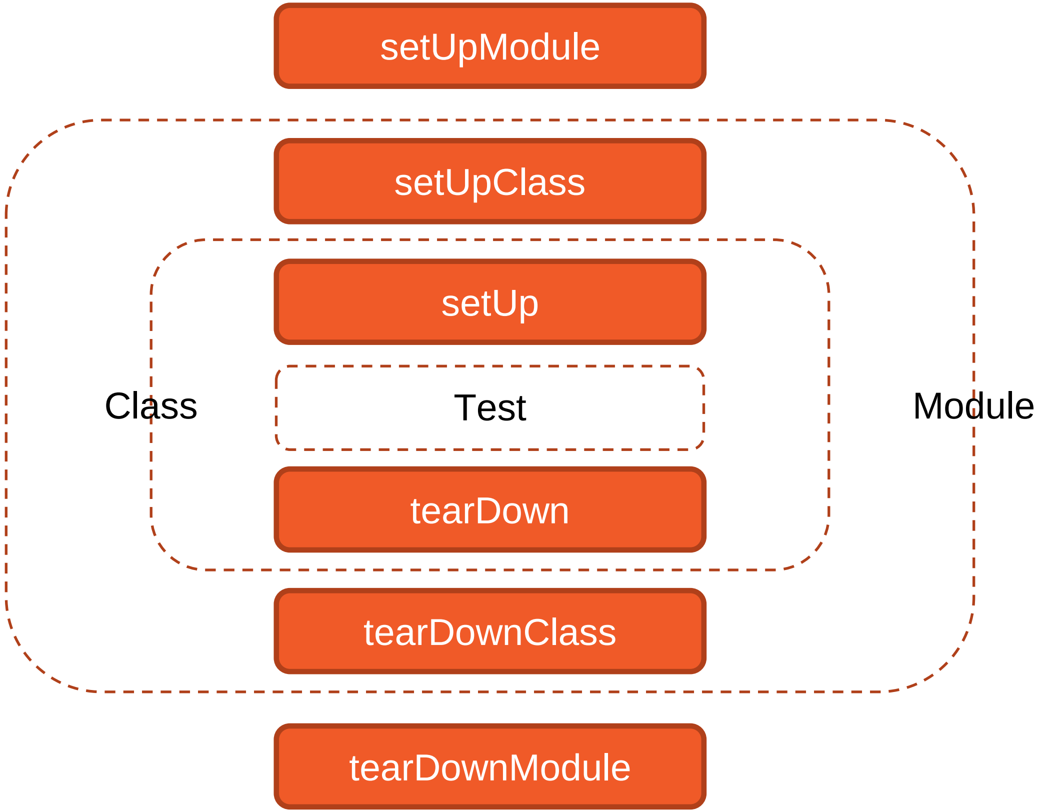
Create a test module and include tests to successfully exercise each type of assert using simple expressions. Investigate/drilldown into documentation on each type of assert as needed to understand its usage.

unittest Framework – Test Fixtures

Rather than repeat common logic across tests, unittest provides a couple of options for capturing:

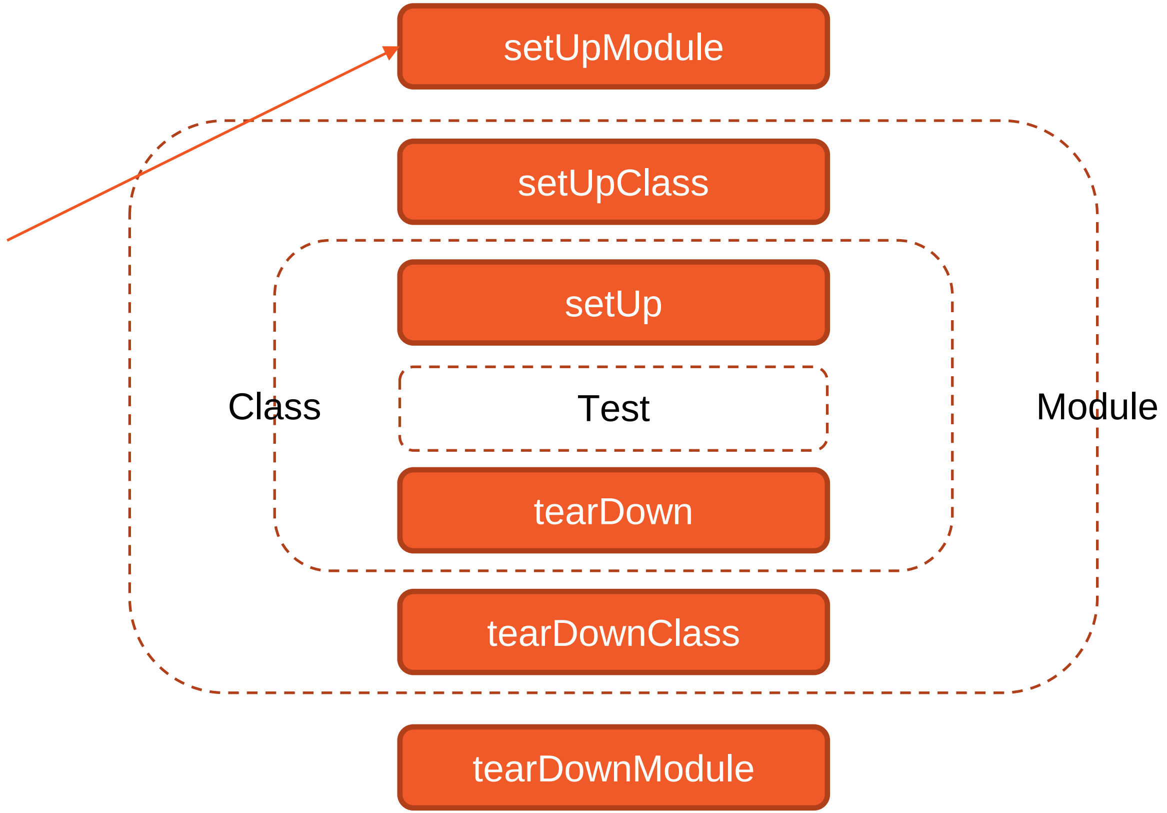
- setUp
- tearDown
- Includes a few different forms

unittest Framework – Test Fixtures

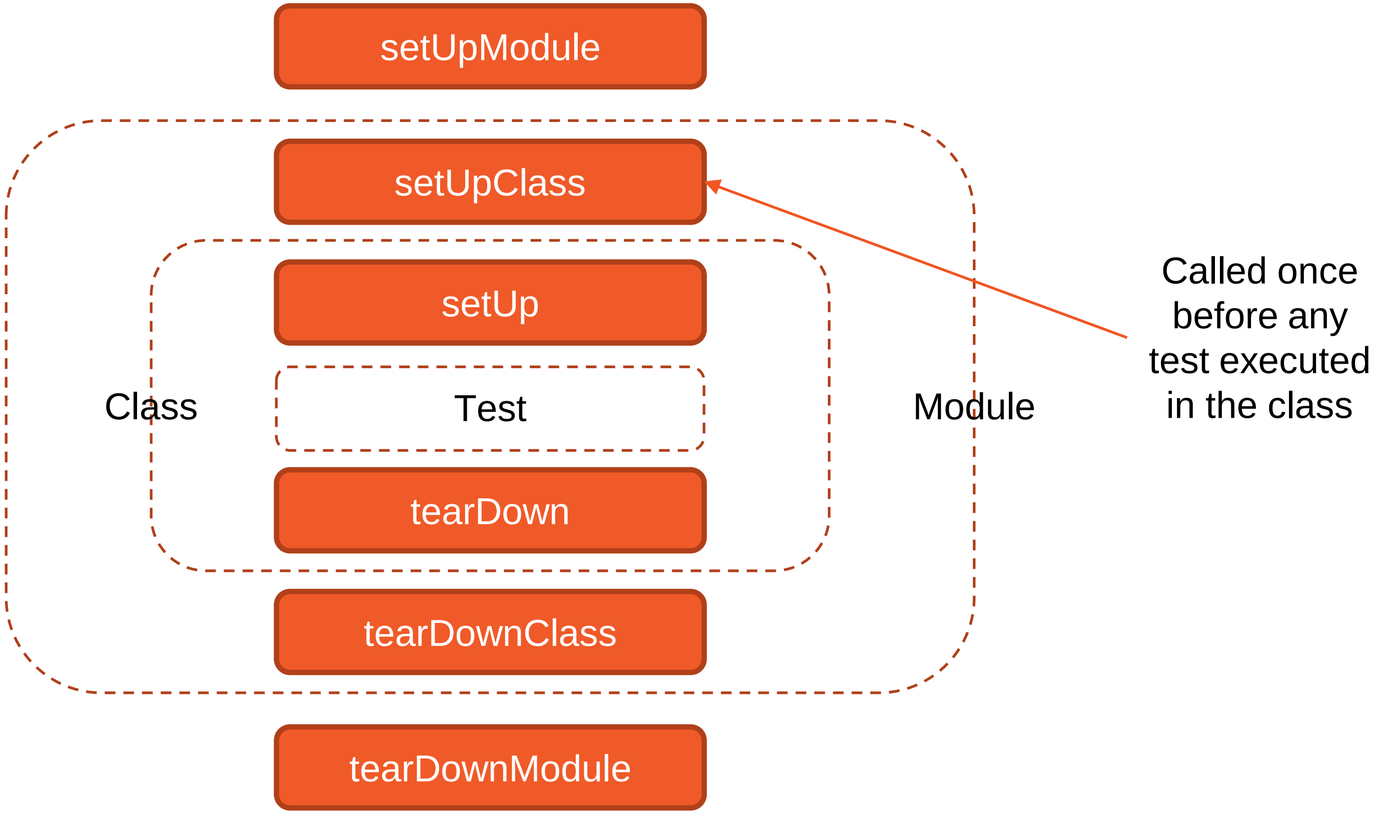


unittest Framework – Test Fixtures

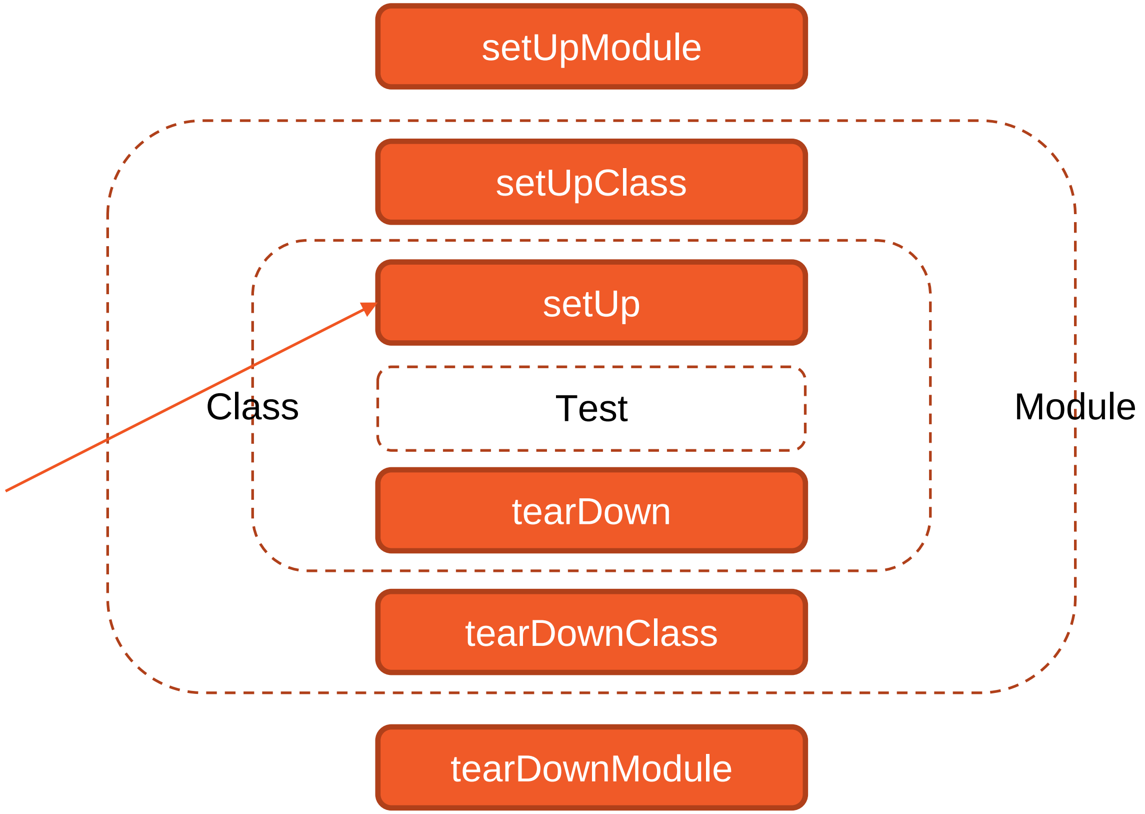
Called once before anything else in the module



unittest Framework – Test Fixtures

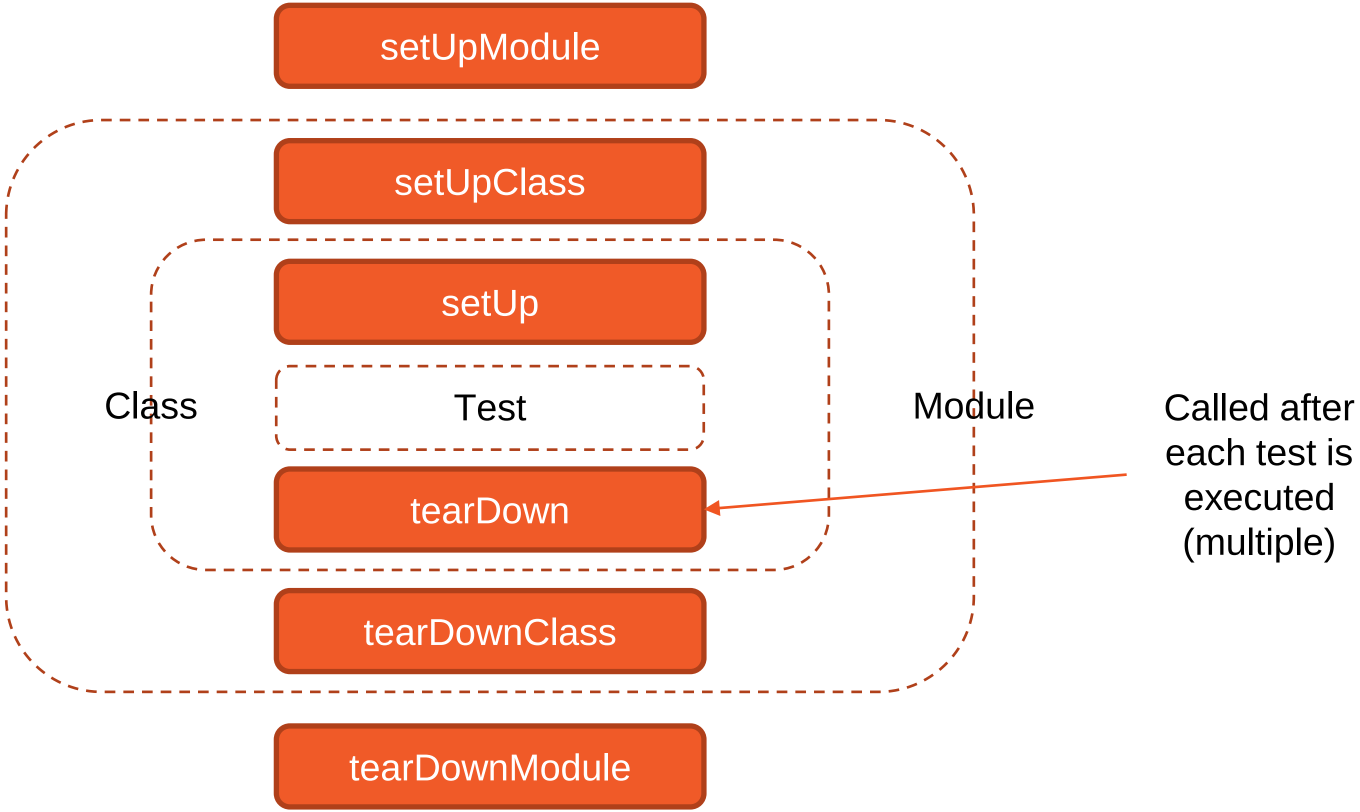


unittest Framework – Test Fixtures

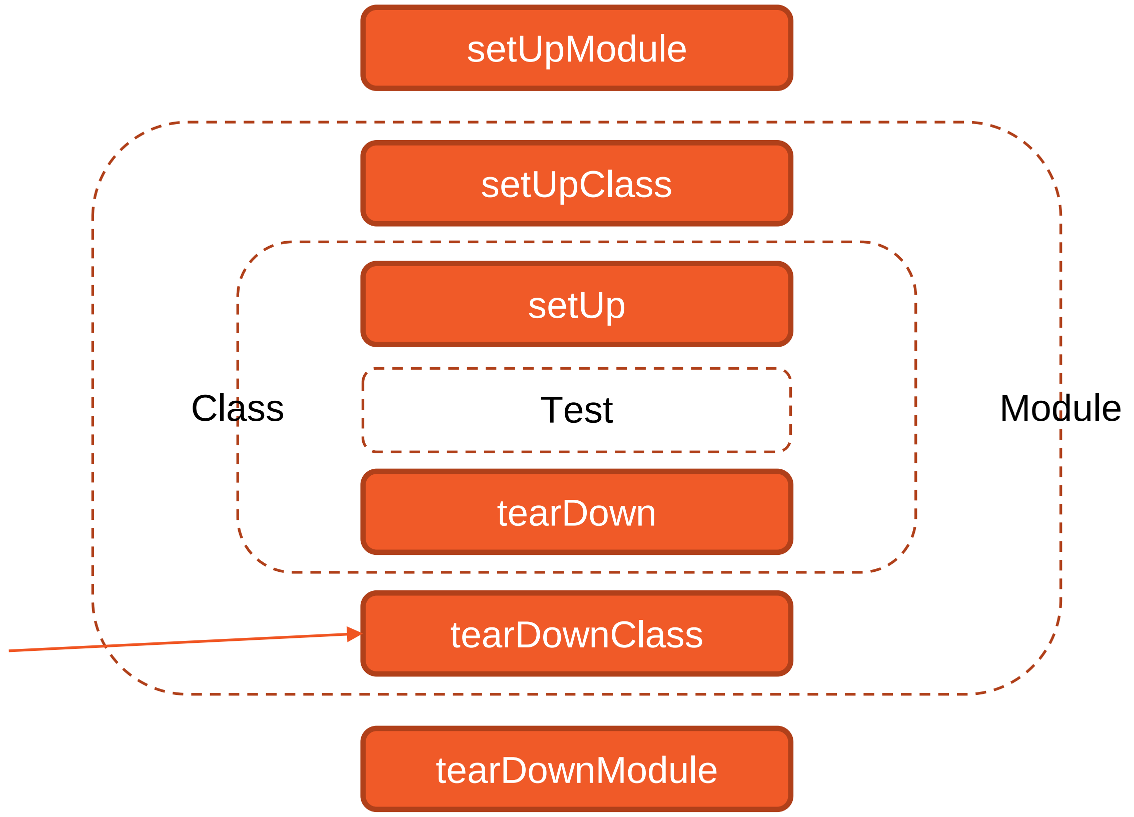


Called before each test is executed (multiple)

unittest Framework – Test Fixtures

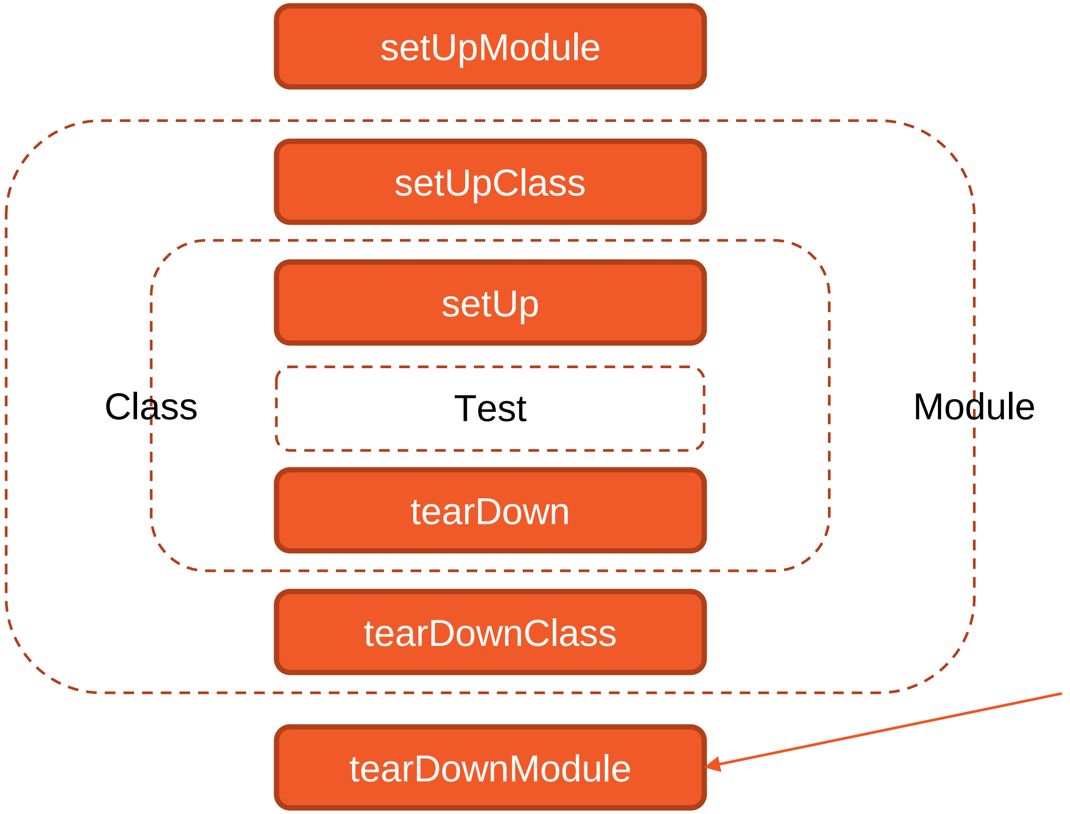


unittest Framework – Test Fixtures



Called once after all tests executed (if setUpClass successful)

unittest Framework – Test Fixtures



Called once after everything else in the module

LAB:

unittest Test Fixtures

Review the documentation on the different types of test fixture methods available with unittest (at https://docs.python.org/3/library/unittest.html#organizing-test-code and https://docs.python.org/3/library/unittest.html#setupclass-and-teardownclass).

Create a test module and include tests to practice utilization of setUp(), tearDown(), setUpClass(), and tearDownClass(). You can use simple print statements to experiment with the different test fixture methods and their timings.

unittest Framework – Managing Test Execution

Options

As an executable module

Using unittest to execute entire module

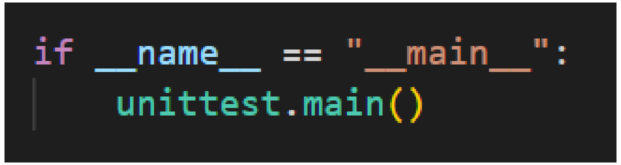
Using unittest to execute single test class (or individual test cases)

Create a test package

unittest Framework – Managing Test Execution

As an executable module

Add the following to your test module:



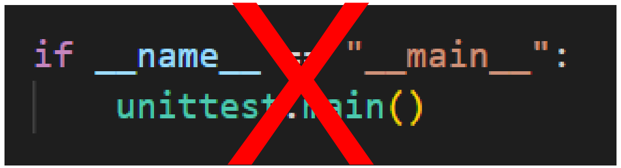
```
if __name__ == "__main__":
    unittest.main()
```

Can execute all tests in module using `python -m <test_module_name>` or `python <test_module_name>.py`

unittest Framework – Managing Test Execution

Using unittest to execute entire module

Does not require:



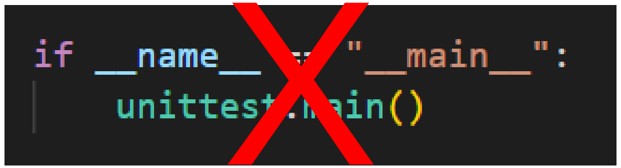
```
if __name__ == "__main__":
    unittest.main()
```

Can execute all tests in module using `python -m unittest <test_module_name>` or `python -m unittest <test_module_name>.py`

unittest Framework – Managing Test Execution

Using unittest to execute single test class (or individual test cases)

Does not require:



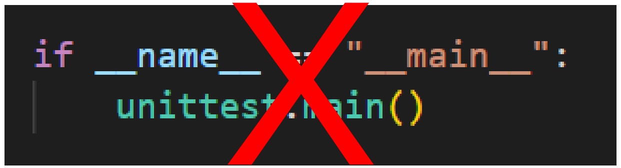
```
if __name__ == "__main__":
    unittest.main()
```

Can execute a specific test class using `python -m unittest <test_module_name>.<test_class_name>`. Can execute a specific test case using `python -m unittest <test_module_name>.<test_class_name>.<test_case_name>`

unittest Framework – Managing Test Execution

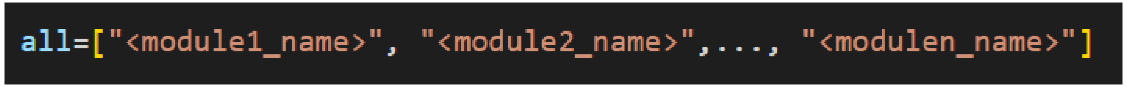
Create a test package

Does not require:



```python
if __name__ == "__main__":
    unittest.main()
```

Capture test modules under a folder and add a `__init__.py` package initializer with simple contents:



```python
all=["<module1_name>", "<module2_name>",..., "<modulen_name>"]
```

From root folder containing package, can execute using variants of `python -m unittest <package_name>.<test_module_name>[.<test_class_name>.<test_case_name>]`

unittest Framework – Test Discovery

unittest also supports a “discovery” mode

- Quickly execute all tests in the project directory and its subdirectories
- Available using `python -m unittest discover` or simply `python -m unittest`
- All test files must be modules or packages importable from top-level directory
- By default, starts from current directory
- By default, searches for test*.py pattern in filenames to identify tests to be executed

unittest Framework – Skipping Tests

While permanently skipping tests can be a “smell”, unittest offers options for temporarily skipping:

- @unittest.skip() decorator
- @unittest.skipIf() decorator
- @unittest.skipUnless() decorator
- self.skipTest method – call within test to initiate a skip

LAB:

unittest – Managing Test Execution

Create a test package containing multiple test modules with tests for simple expressions/asserts. Experiment with the various ways you can execute different combinations of tests.

Review the documentation on the different types of basic skip options available with unittest (at https://docs.python.org/3/library/unittest.html#skipping-tests-and-expected-failures). Create a test module and include tests to practice utilization of each of the basic types (skip, skipIf, skipUnless, and self.skipTest).

unittest Framework – Managing Test Failures

Sometimes, our tests fail

- Any Exception raised in a test case causes that test case to fail
- The assert methods raise exceptions if the expected result does not occur (i.e., assertEqual determines not equal)
- For certain edge or exception cases, we may expect an Exception to be raised
- In those cases, an Exception is success – tests that our code correctly handles bogus data
- assertRaises is provided by the unittest framework for this purpose

LAB:

unittest – Testing Classes

Build a set of Python classes and unit tests as described in the lab detail at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/unittest-python-movie.

LAB:

unittest – Testing Classes

Build a set of Python classes and unit tests as described in the lab detail at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/unittest-python-wordcount.

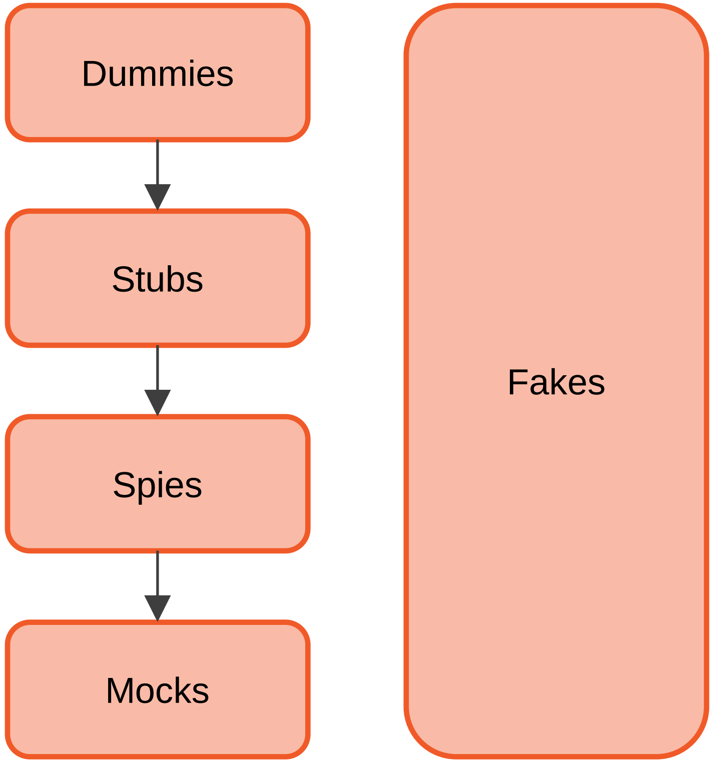
Mocking – Testing Components in Isolation

Mocking / Test Doubles

We use mocking/test doubles:

- To help isolate our testing to focused areas of code
- To manage dependencies and help control testing against them
- To force special cases that exercise specific code flows
- To bring stability and consistency to our tests
- Helps make the “unit” in unit testing possible

Mocking / Test Doubles



Dummies

- Simplest form of test double
- No real functionality – mostly a placeholder
- When dependency exists but is not needed for the test
- Create and use an empty or straightforward copy
- Returns meaningless value on every call

Stubs

- Same response regardless of parameters passed
- Used to test different paths through code
- Allows you to control the value of the dependency
- Can also use to force an exception
- Create a copy of the dependency and return same value

Spies

- Can return a value like a stub
- Also provides info about how called
  - ➢ Verify member call with specific parameter values
  - ➢ Verify member call a specific number of times
  - ➢ Verify whether member called or not
- Helpful with testing at third party boundaries
- Helpful with managing side effects of dependencies

Mocks

- Programmable spies
- Can return a set value like a spy
- Difference is ability to setup test data in the double
- Provides additional flexibility - can code the setup
- Still only returns set value but it’s the value coded
- Trade-off – requires setup work

Fakes

- Most powerful type of test double
- Acts like the class/component it’s doubling for
- Double that’s more sophisticated (but still controlled)
- Useful for managing downstream dependencies
    - Gating database connections
    - Gating web service calls
    - File operations
- Can appear to integrate but control results

DEMO:

unittest – Mocking

Coverage

Coverage

- Coverage in unit testing measures the % of your code under test that is exercised by a unit test
- Coverage value can range between 0 and 100 %
- Question: What is the right target % of coverage to aim for?

Coverage

- coverage.py can be used to measure test coverage in your Python app
- Install using pip (*pip install coverage*)
- Use *coverage run* to measure
- For unittest, use *coverage run -m unittest <target>*

Coverage – Measuring Results

- Use *coverage report* to report coverage results
- Alternatively, coverage html can be used to view coverage results in a more user-friendly manner

LAB:

unittest – TDD and Mocking

Walk through the notebook (including code examples and exercises) at https://github.com/KernelGamut32/intermediate-python/blob/main/15%20Testing%20%2B%20TDD%20%2B%20Mocking.ipynb.

LAB:

unittest – TDD and Mocking

Build a set of Python classes and unit tests as described in the lab detail at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/unittest-python-mocking.

Docker & Kubernetes

Why Containerization?

- Abstracts away the code implementation so you can deploy in a platform-agnostic manner, writing in the language of your choice
- Aligns strongly with the principles and practices of DevOps
- Helps leverage the power of the cloud
- Speeds up important non-coding activities (infrastructure spin-up, testing, CI/CD tasks, DevSecOps, code quality checks, etc.
- Helps breed consistency vs. “snowflake”

What Do We Mean by Application Hosting?

- Target infrastructure and runtime platform used for deployment and execution
- Can include compute (CPU and server resources), storage, network, data, and operating system

What Are the Hosting Options with Cloud?

- IaaS (Infrastructure-as-a-Service)
- PaaS (Platform-as-a-Service)
- Serverless / FaaS (Functions-as-a-Service)
- SaaS (Software-as-a-Service)
- Containers

Containers

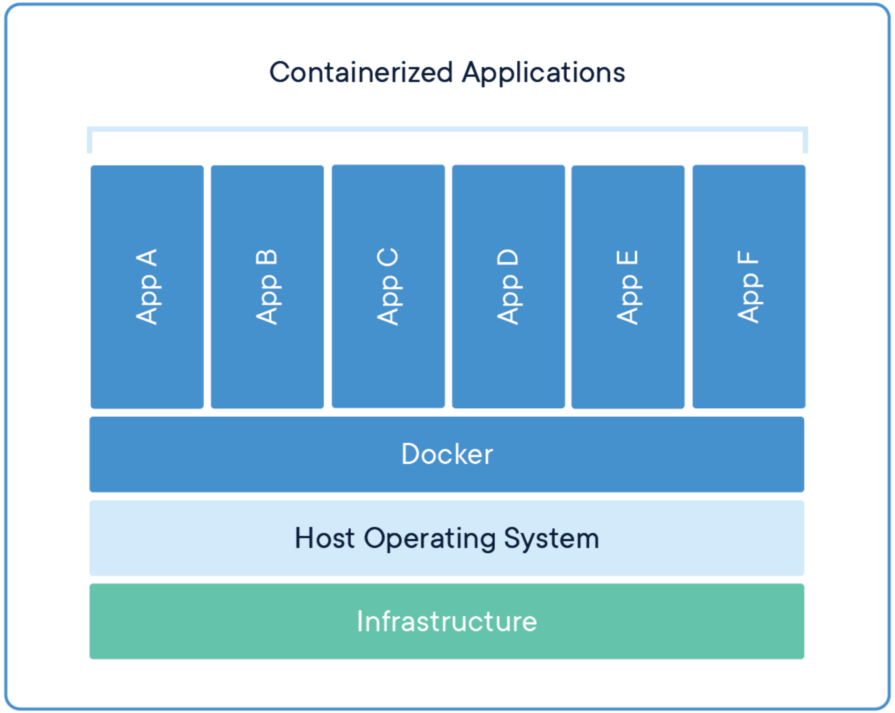
- Form of virtualization at the app packaging level (like virtual machines at the server level)
- Isolated from one another at the OS process layer (vs VM’s which are isolated at the hardware abstraction layer)
- Images represent the packaging up of an application and its dependencies as a complete, deployable unit of execution (code, runtime and configuration)

Containers

- A platform (e.g., Docker) running on a system can be used to dynamically create containers (executable instances of the app) from the defined image
- Typically, much, much smaller than a VM which makes them lightweight, quickly deployable and quick to “boot up”
- An orchestration engine (e.g., Kubernetes) might be used to coordinate multiple instances of the same container (or a “pod” of containers) to enable the servicing of more concurrent requests (scalability)

docker + kubernetes

Containers


Containerized Applications
App A
App B
App C
App D
App E
App F
Docker
Host Operating System
Infrastructure

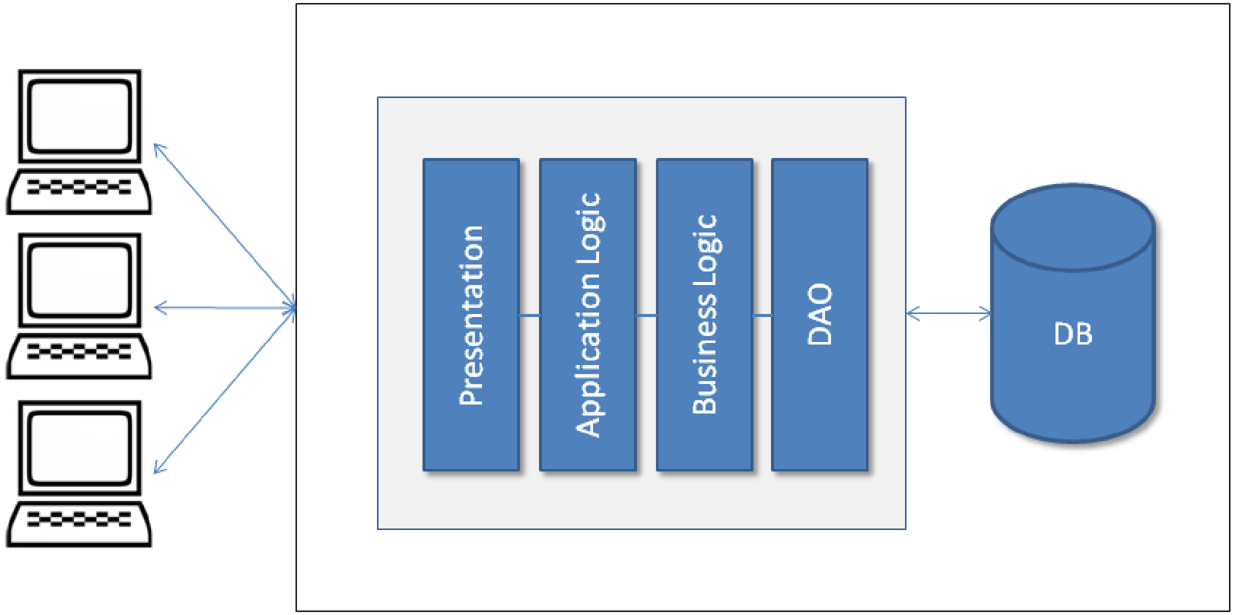
What are Microservices?

- An architectural style in which a distributed application is created as a collection of services that are:
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable (by extension, independently scalable)
  - Domain centric and organized around business capabilities
- The Microservices architecture enables the continuous deployment and delivery of large, complex distributed applications

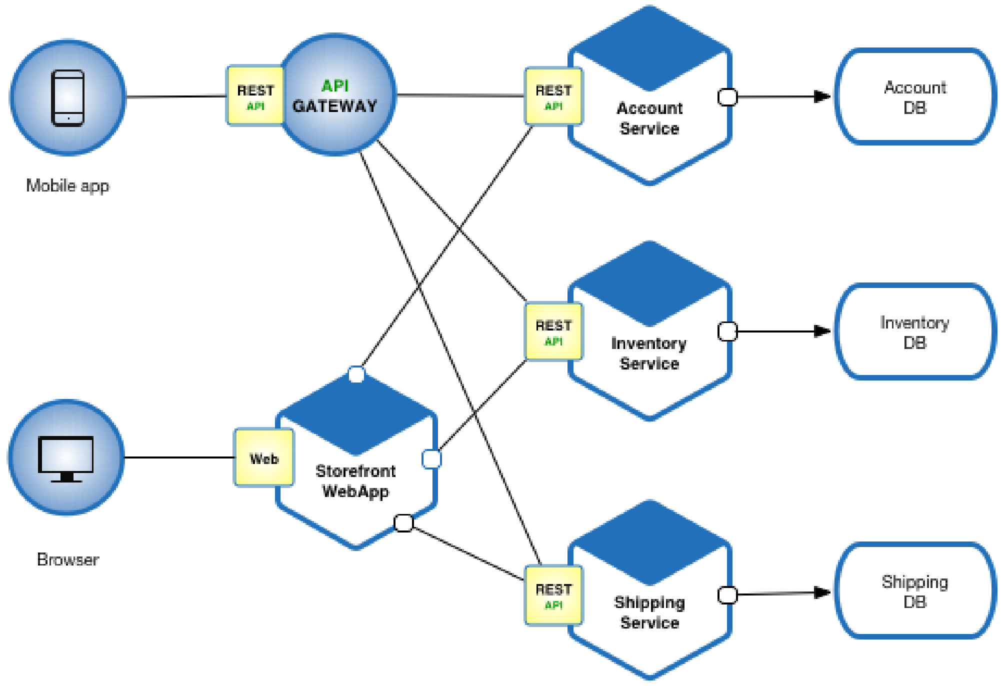
What is a Monolith?

- Typical enterprise application
- Large codebases
- Set technology stack
- Highly coupled elements
- Whole system affected by failures
- Scaling requires the duplication of the entire app
- Minor changes often require full rebuild

Monolithic Architecture Example



Example Microservices Architecture



Microservices – Benefits vs. Costs

Benefits:

- Enables work in parallel
- Promotes organization according to business domain
- Advantages from isolation
- Flexible in terms of deployment and scale
- Flexible in terms of technology

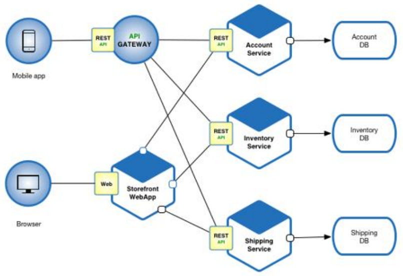
Microservices – Benefits vs. Costs

Costs:

- Requires a different way of thinking
- Complexity moves to the integration layer
- Organization needs to be able to support re-org according to business domain (instead of technology domain)
- With an increased reliance on the network, you may encounter latency and failures at the network layer
- Transactions must be handled differently (across service boundaries)

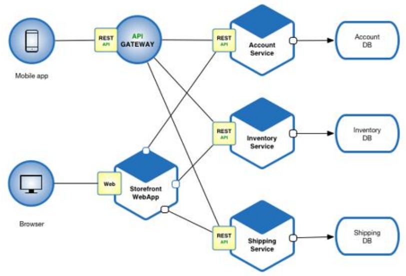
Microservices & Containers

- Microservices – with their smaller size, independently-deployable and independently-scalable profile, and encapsulated business domain boundary – are a great fit for containers
- Using Kubernetes, sophisticated systems of integrated microservices can be built, tested and deployed
- Leveraging the scheduling and scalability benefits of Kubernetes can help an organization target scaling across a complex workflow in very granular ways
- This helps with cost management as you can toggle individual parts of the system for optimized performance



Microservices & the Cloud

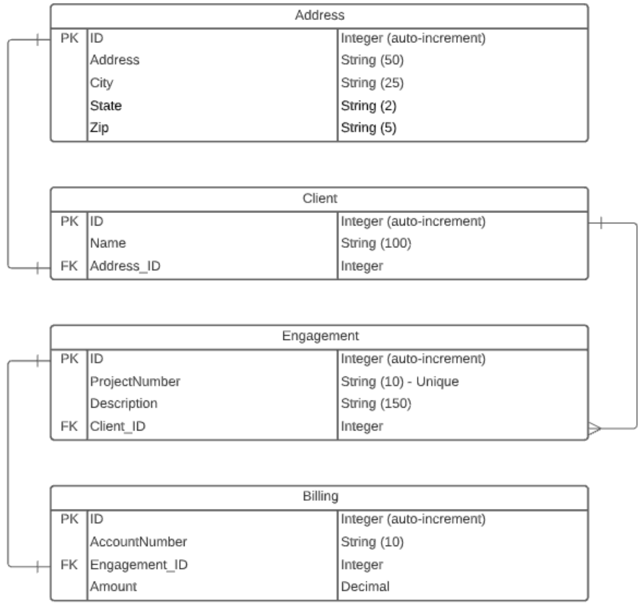
- Microservices have broad support across multiple Cloud providers
- One option includes standing up VM’s (IaaS) and installing / managing a Kubernetes cluster on those machines or
- Another option includes leveraging a managed service (PaaS) provided by the CSP
- Microservices are a great option for the Cloud because the elastic scalability provided by the Cloud infrastructure can directly support the independent scalability needed with a microservices architecture



Microservices Summary

- Applications that can
  - ➢ Efficiently scale
  - ➢ Are flexible
  - ➢ Are high performance
- These apps are powered by multiple services, working in concert
- Each service is small and has a single focus
- We use a lightweight communication mechanism to coordinate the overall system
- This ends up giving us a technology agnostic API

Use Case Discussion – Monolith to Microservices

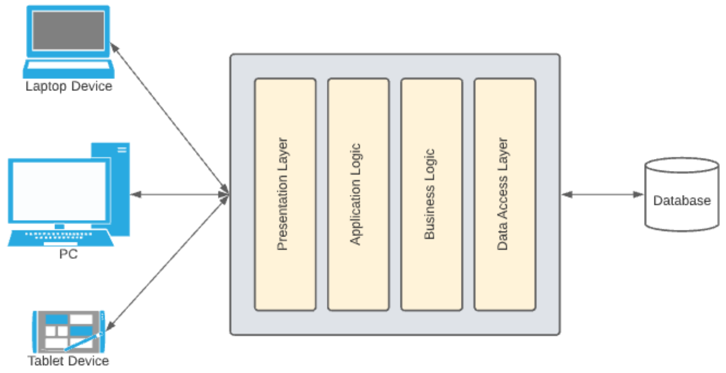


Address		
PK	ID	Integer (auto-increment)
	Address	String (50)
	City	String (25)
	State	String (2)
	Zip	String (5)

Client		
PK	ID	Integer (auto-increment)
	Name	String (100)
FK	Address_ID	Integer

Engagement		
PK	ID	Integer (auto-increment)
	ProjectNumber	String (10) - Unique
	Description	String (150)
FK	Client_ID	Integer

Billing		
PK	ID	Integer (auto-increment)
	AccountNumber	String (10)
FK	Engagement_ID	Integer
	Amount	Decimal



Operations Supported:

- Add a new client (with address)
- Update an existing client (with address)
- "Soft" delete a client (with address)

- Add a new engagement associated to a client
- Update an existing engagement associated to a client
- "Soft" delete an engagement - client remains unchanged

- Add a new billing entry associated to an engagement
- Updates and deletes are not allowed - to adjust, apply compensating transactions as required

- Ability to view all entities - list of customers (with address), list of engagements for a client, and list of billing entries for an engagement
- Using the system, users can initiate the generation of a batch report - this operation will generate and send a file to the presentation layer for formatted display

Group Discussion:

Microservices

In your assigned breakout rooms, using the architecture diagram for current state of the client engagement system (on the previous slide) and using the information on the previous set of slides on containerization and microservices, discuss the following as a team:

- Where might you apply containerization in the implementation of an application rewrite?
- What are some ways you could apply the microservices pattern as part of an application rewrite?
- What are some specific benefits you might offer to justify the containerization and microservices investment?
- What are some specific caveats or potential challenges that you would want to help the project team and business keep in mind with moving in this direction architecturally?

Nominate someone from your team to share the results of your team’s discussion.

Modern Approaches to Application Development

- Leverage the cloud
- Abstract away as much of the non-code elements as possible
- Use the tools that arise from DevOps (automation, feedback, etc.)
- Integrate testing and code quality checks early (“Shift Left”)

Design Approaches to Modern Applications

- Decompose when possible
- Embrace Failure
- Separate the language from the function of the code

Infrastructure as Code (IaC)

- Declarative vs. Imperative coding
- Drawbacks of declarative infrastructure
- Impact of IaC on the dev process, on testing, and on deployment

What is Docker?

- Built around the concept of images & containers
- Also, supports composition of a set of containers to be deployed together
- For example, application code + network components + database components

What is Docker?

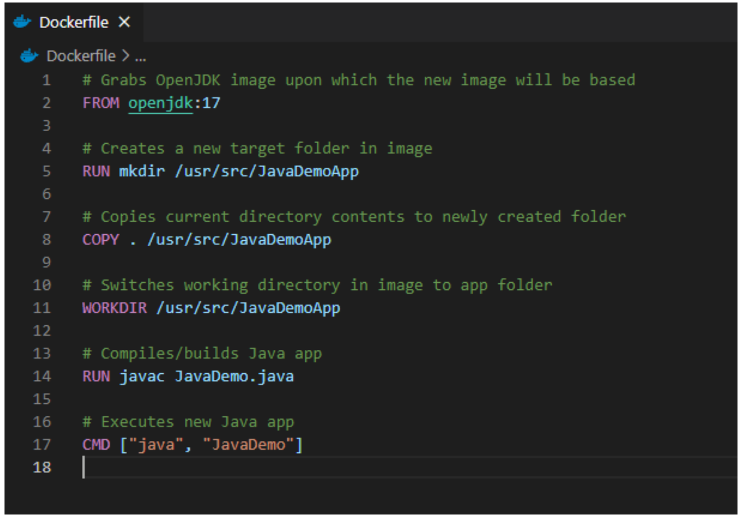
- Utilizes principles of “immutable infrastructure”
- Complete application environments torn down and recreated as needed
- Helps to minimize infrastructure “drift” and environment inconsistencies

The Dockerfile

- Tells Docker what to do in creating an image for your application
- The commands are all things you could do from the CLI
- Used by the docker “build” command
- Docker build uses this file and a “context” – a set of files at a specified location – to make your image

Dockerfile example

The following creates an image for building/running Java app in container
See https://github.com/KernelGamut32/dockerlab-repo-sample for sample



Dockerfile ×

Dockerfile > ...

```dockerfile
# Grabs OpenJDK image upon which the new image will be based
FROM openjdk:17

# Creates a new target folder in image
RUN mkdir /usr/src/JavaDemoApp

# Copies current directory contents to newly created folder
COPY . /usr/src/JavaDemoApp

# Switches working directory in image to app folder
WORKDIR /usr/src/JavaDemoApp

# Compiles/builds Java app
RUN javac JavaDemo.java

# Executes new Java app
CMD ["java", "JavaDemo"]

```

Docker Images

- Represent templates defining an application environment
- New instances of the application can be created from the image
- These instances are called containers

Docker Images

- Images are defined via a Dockerfile definition
- Support layers for building up the environment in stages
- Fully defines the application, including all components required to support

Docker Images

- Those components can include:
    - ➢ Runtime
    - ➢ Development framework
    - ➢ Source code
    - ➢ Executable instructions for container startup

Docker Images

- Start with a base that gives your app a place to live
  - Needed OS/runtimes/dB server applications, etc.
- Examples:
  - nginx
  - Node
  - MySQL
  - Apache HTTP Server
  - IIS with .NET Runtimes

Docker Hub

- Centralized registry for image storage & sharing
- Can signup for an account – user accounts offer both free and pro versions
- Also, supports organizations for grouping of multiple team members

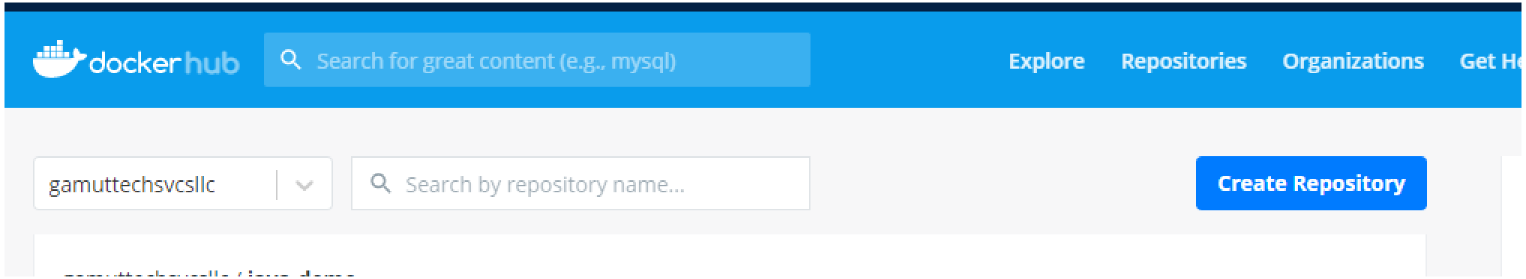
Docker Hub

- Accessible at https://hub.docker.com
- Search feature enables search for image by technology or keyword
- Image detail displays available tags and image variants

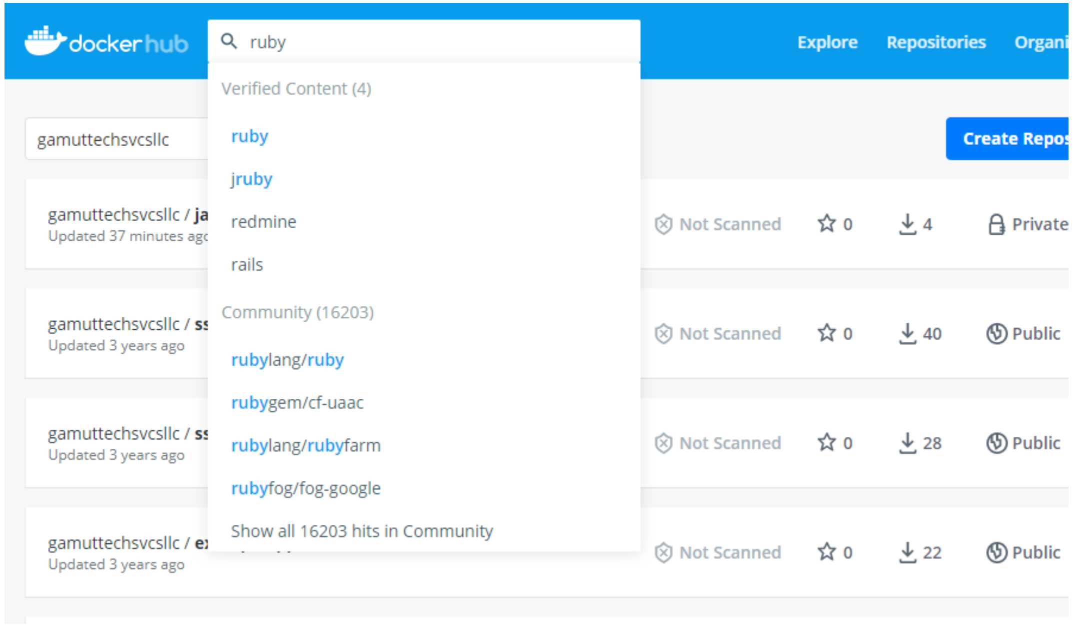
Docker Hub

- May also include examples of usage
- Pro account supports image scanning for security vulnerabilities
- Can be useful for image reuse and image sharing across a dev team
- There are other registry types – including private registries

Docker Hub


docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Get He
gamuttechsvcsllc
Search by repository name...
Create Repository

Docker Hub



Verified Content (4)

ruby

jruby

redmine

rails

Community (16203)

rubylang/ruby

rubygem/cf-uaac

rubylang/rubyfarm

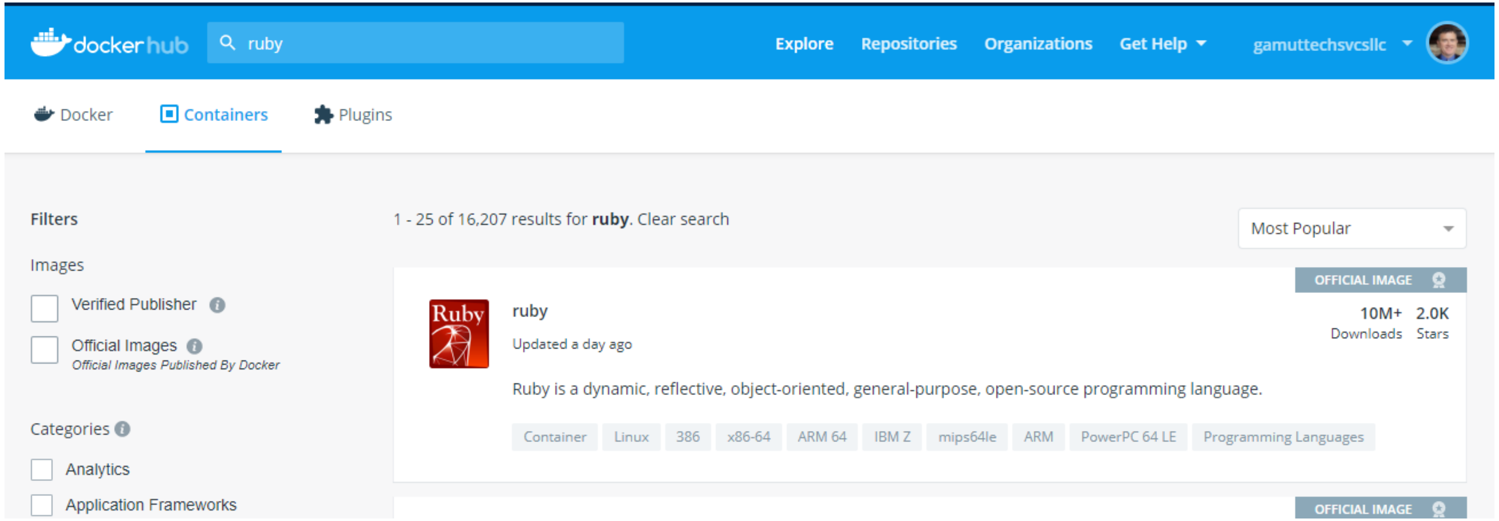
rubyfog/fog-google

Show all 16203 hits in Community

gamuttechsvcsllc

gamuttechsvcsllc / ja Updated 37 minutes ago	Not Scanned	0	4	Private
gamuttechsvcsllc / ss Updated 3 years ago	Not Scanned	0	40	Public
gamuttechsvcsllc / ss Updated 3 years ago	Not Scanned	0	28	Public
gamuttechsvcsllc / ex Updated 3 years ago	Not Scanned	0	22	Public

Docker Hub



Filters

Images

Verified Publisher

Official Images
Official Images Published By Docker

Categories

Analytics

Application Frameworks

1 - 25 of 16,207 results for **ruby**. Clear search

Most Popular

OFFICIAL IMAGE

ruby

Updated a day ago

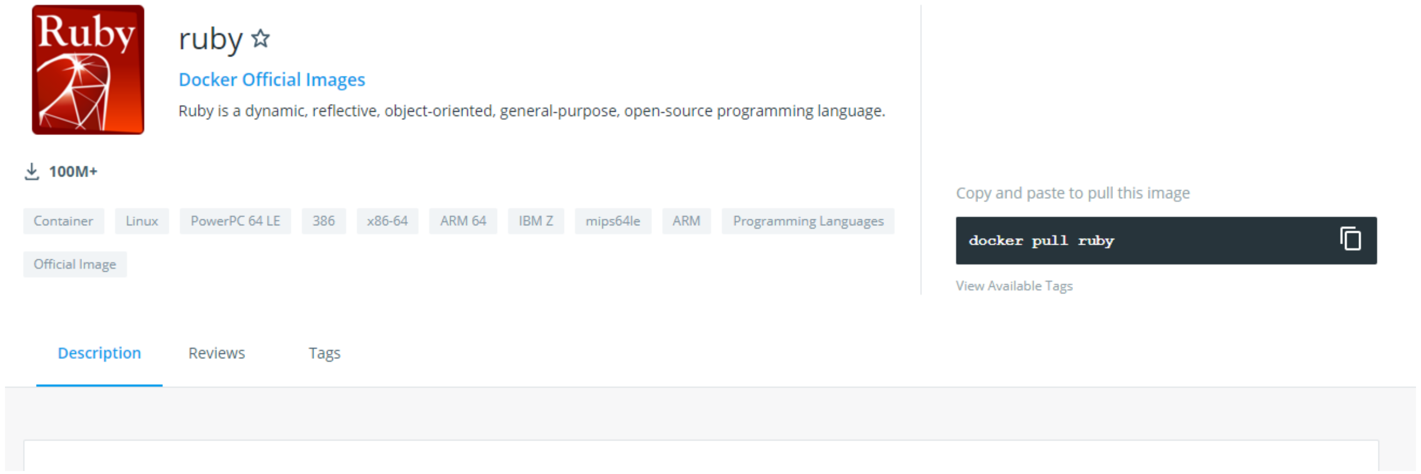
10M+ Downloads 2.0K Stars

Ruby is a dynamic, reflective, object-oriented, general-purpose, open-source programming language.

Container | Linux | 386 | x86-64 | ARM 64 | IBM Z | mips64le | ARM | PowerPC 64 LE | Programming Languages

OFFICIAL IMAGE

Docker Hub


ruby ☆

Docker Official Images

Ruby is a dynamic, reflective, object-oriented, general-purpose, open-source programming language.

100M+

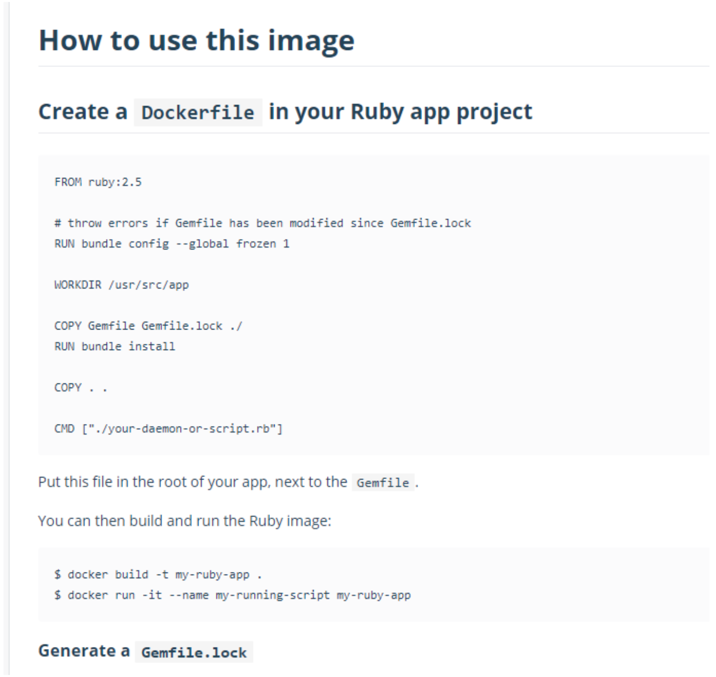
Container | Linux | PowerPC 64 LE | 386 | x86-64 | ARM 64 | IBM Z | mips64le | ARM | Programming Languages | Official Image

Copy and paste to pull this image

```
docker pull ruby
```

View Available Tags

Docker Hub



How to use this image

Create a `Dockerfile` in your Ruby app project

```
FROM ruby:2.5

# throw errors if Gemfile has been modified since Gemfile.lock
RUN bundle config --global frozen 1

WORKDIR /usr/src/app

COPY Gemfile Gemfile.lock ./
RUN bundle install

COPY . .

CMD ["./your-daemon-or-script.rb"]
```

Put this file in the root of your app, next to the `Gemfile`.

You can then build and run the Ruby image:

```
$ docker build -t my-ruby-app .
$ docker run -it --name my-running-script my-ruby-app
```

Generate a `Gemfile.lock`

Building The Image

- To build the image from Dockerfile use *docker build*
- *docker build -t <tag name> <path to Dockerfile>*
- For example, *docker build -t java-demo .*
- Builds image from Dockerfile in current folder (.) with tag name “java-demo”

Building The Image

- For tag name, can include optional detail:
  - ➢ Docker ID in Docker Hub for eventual push to image registry
  - ➢ Version identifier for tag – defaults to “latest” if excluded
- For example, *docker build -t <docker ID>/<tag name>:<version> .*

Docker Review Questions

- *What is the immediate result of a Docker build?*
- *A – Container*
- *B – Image*
- *C - Cluster*

Docker Review Questions

- *What do we use to create a Docker image?*
- *A – Dockerfile*
- *B – Docker Image*
- *C - Dockerhub*

Docker Review Questions

- *Where do Docker images get shared?*
- *A – In a registry*
- *B – In Dockerhub*
- *C - In the Docker store*

Docker Review Questions

Which of the following is a cost/disadvantage of microservices?

- *A – We can use Java with one microservice, and Python with a different microservice*
- *B – Isolation of microservices*
- *C – Increased dependence on the network*

LAB:

Docker

Execute the 2 parts of the Docker lab available at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/docker-lab.

Application Bootstrapping with Docker and k8s

- Kubernetes provides a hosting environment for containerized applications
- Once you have a Docker image, you can work entirely within Kubernetes to deploy your app

Kubernetes (k8s) Overview

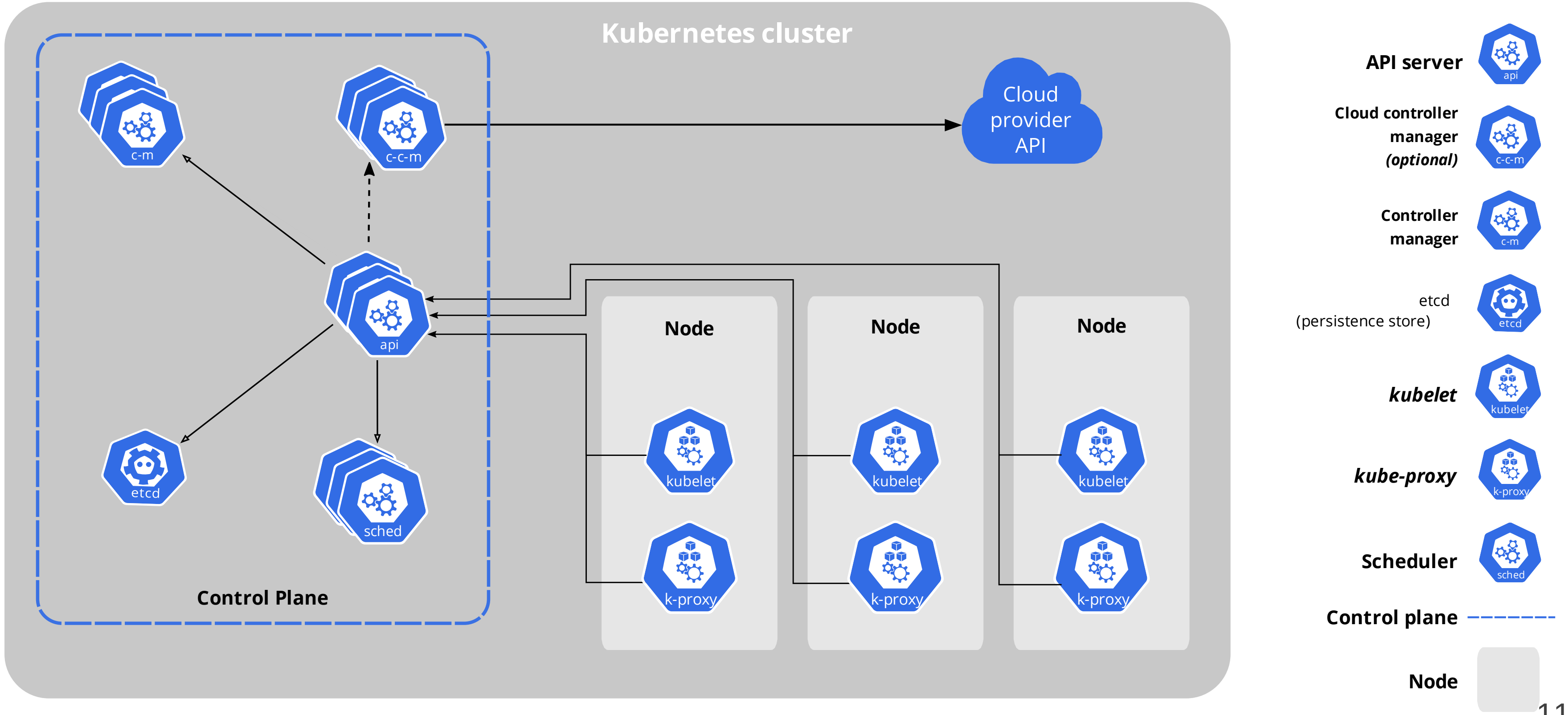
- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services

What is an Orchestrator and Why Do We Need It?

kubernetes

- What would it look like to manually control the containers needed for your application as your app scales or containers fail?
- “Orchestration” is the execution of a defined workflow
- We can use an orchestrator to start and stop containers automatically based on your set rules
- What does this open up for you?

Architecture of k8s System



Core Components of k8s

- When you deploy Kubernetes, you get a cluster.
- A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.
- The worker node(s) host the Pods that are the components of the application workload.
- The control plane manages the worker nodes and the Pods in the cluster.

k8s Resource/Manifest

- Kubernetes manifests are text files, usually written in YAML
- They are used to create, modify and delete Kubernetes resources

Kubernetes Control Plane & Data Plane

kubernetes

- Kubernetes is actually a collection of pods implementing the k8s api and the cluster orchestration logic (AKA the Kubernetes control plane)
- The application/data plane is everything else, meaning all the nodes that host all the pods that the application runs on
- The *controllers* of the control plane implement control loops that repeatedly compare the desired state of the cluster to its actual state
- When the state of the cluster changes, controllers take action to bring it back inline with desired state

Declarative and Desired State

- Kubernetes supports a declarative model – we can send the api a yaml file in a declarative form
- Desired state means that we specify in the yaml file our desired state and the cluster takes responsibility to make sure it will happen
- We describe the desired state using a yaml or json file that serves as a record of intent, but we do not specify how to get there (this is kubernetes responsibility to get us there)
- Things could change or go wrong over the lifetime of the cluster (node failing, etc…), Kubernetes is responsible to always make sure that the desired state is kept intact
- Kubernetes control plane controllers are always running in a loop and checking that the actual state of the cluster matches the desired state, so that if any error occurs they kick in and rectify the cluster

Reconciling state

- Watch for the spec fields in the YAML files
- The *spec* describes *what we want the thing to be*
- Kubernetes will *reconcile* the current state with the spec (technically, this is done by a number of *controllers*)
- When we want to change a resource, we update the *spec*, and reapply
- Kubernetes will then *converge* that resource

k8s Pods: The Basic Building Block

kubernetes

- A pod represents a set of running containers in your cluster
- Worker nodes host pods
- The control plane manages the worker nodes and the pods inside them

k8s Pods: The Basic Building Block

- Pod is the basic execution and scaling unit in Kubernetes
- Kubernetes runs containers but always inside pods
- It is like a sandbox in which containers run (abstraction)
- A pod groups containers for execution together on same node, sharing resources (RAM, CPU, network, volumes, etc.)

Pod state

- Pods are considered to be relatively ephemeral (rather than durable) entities
- Pods do not hold state, so if a pod crashes, Kubernetes will replace it with another
- Multiple instances of the same pod are called replicas

Deployments

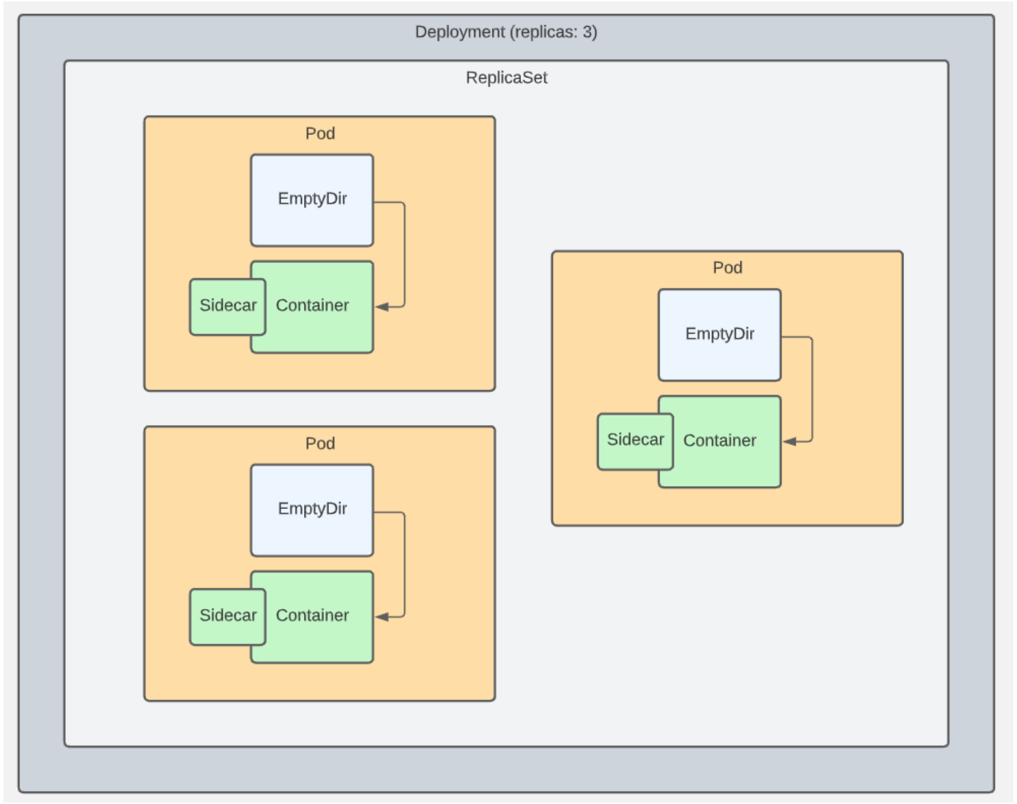
- A *Deployment* controller provides declarative updates for Pods and ReplicaSets
- The Deployment Object gives us a better way of handling the scaling of pods
- The advantage of using Deployment versus using a replicaset is having rolling updates support for the pod container versions out-of-the-box

Deployments

- Every time the application code changes, a new version of the application container is built, and then there is a need to update the Deployment manifest with the new version and tell K8s to apply the changes
- K8s will then handle the rolling-out of this newer version, terminating pods with the old version as it spins up the new pods with the updated container
- This means that at some point we will have multiple versions of the same application running at the same time

kubernetes

Pods, Deployments, & ReplicaSets



LAB:

Kubernetes – Deployments

Execute the lab available at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/k8s-deployments.

Question for the Group

This component is used in a Kubernetes cluster to store state about the cluster and the resources it is responsible for:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Question for the Group

This component is used in a Kubernetes cluster to monitor current state against desired state, and notify the cluster when an update is required to resolve any gaps:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Question for the Group

This component provides the programmatic interface to the Kubernetes cluster and is used by other components and users for retrieving data about the cluster and applying changes to the cluster:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Addons: Cluster DNS

- A DNS system that runs in a pod in your cluster
- It also spins up a k8s Service
- Allows the use of consistent DNS names instead of IP addresses

k8s Probes

- The pods in a node are not always up and running, ready for incoming traffic
- We need a way to check
- The kubelet does this through periodic “liveness” checks into the pods in its node

kubectl

- kubectl (“kube-cuttle”) is command-line tool used to interact with API
- May require installation (automatic if using k8s via Docker Desktop)
- Config file defines details of connection to cluster (e.g., ~/.kube/config)
- Run `*kubectl cluster-info*` to confirm connectivity
- Run `*kubectl*` from command-line (with arguments) to see options
- NOTE: k8s in Docker Desktop should handle most of this for you

Namespaces in k8s

- Provide a grouping mechanism in Kubernetes
- Every k8s object belongs to a namespace
- Every Cluster automatically includes a default namespace

Namespaces in k8s

- Allow you create logical separation within a physical Cluster
- Provide a boundary for security and resource control
- Can explicitly create a namespace and deploy resources to it using namespace field in manifest or --namespace arg on kubectl

Namespaces in k8s

- Objects within a namespace are isolated from others
- Enables creation of multiple instances of same apps with same names
- Resources from one namespace can communicate with another namespace using Services
- Controller only looks for matching resources in its namespace

Contexts

- Can be used to define connection details for a k8s Cluster
- Sets the default namespace to be used
- Prevents need to specify namespace – will use context to automatically set if excluded (like with default in standard install)
- Able to switch between contexts using use-context command

Contexts

- Can also be used to manage switching between different Clusters (local or remote)
- For remote, API server will be secured with TLS and kubectl uses a client cert for AuthN

LAB:

Kubernetes – Namespaces

Execute the lab available at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/k8s-namespaces.

ConfigMaps vs. Environment Variables

- A ConfigMap is an API object that lets you store configuration for other objects to use
- These are key/value pairs
- This lets you decouple environment-specific config data from your container images, making your apps more portable
- CAUTION: ConfigMaps do not provide encryption or secrecy. Use a Kubernetes Secret for that.

ConfigMaps

- Can be:
  - ➢ Set of key-value pairs
  - ➢ Blurb of text
  - ➢ Binary files
- One pod can use many ConfigMaps and one ConfigMap can be used by many pods
- Data is read-only – pod can't alter

ConfigMaps as Env Vars

- Can be created from a literal:
  `` `kubectl create configmap <name> --from-literal=<key>=<value>` ``
- Can be created from a file (to group together multiple settings):
  `` `kubectl create configmap <name> --from-env-file=<file-path>` ``

Where <file-path> contains .env file like:

```
key1=value1
key2=value2
```

- Can be created from a .yml definition file (like all things k8s)

ConfigMaps

- Can be presented as files inside directories in the container
- Uses volumes – making contents of ConfigMap available to pod
- Uses volume mounts – loads contents of ConfigMap volume into specific container path in pod
- Can contain settings to override defaults

ConfigMaps - Precedence

- If env vars defined in multiple places, definitions in `env` section in pod spec will trump others (localized)
- Typical approach:
  - Default app settings “baked in” to container image (e.g., to support dev mode)
  - Specific settings for an environment stored in ConfigMap and surfaced to container filesystem
  - Merges with default settings (overwriting where applicable)
  - Final tweaks can be accomplished with env variables in pod spec

Secrets in Pods

- Secrets let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.
- A pod needs to reference a secret. There are three ways get to the secret:
- As a file in a Volume that is mounted to a container;
- As an environment variable for a container;
- By the kubelet, when it pulls images for the pod
- NOTE: The default config for a Secret does NOT have encryption – data is plain text. You need to configure "Encryption at Rest" for the Secret", and Role Based Access Control in your cluster

LAB:

Kubernetes – ConfigMaps & Secrets

Execute the lab available at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/k8s-cms-and-secrets.

DEMO:

Kubernetes – ConfigMaps & Secrets

k8s Services

kubernetes

- Services are an abstraction
- They define a set of pods, and an access policy for them
- A pod has an IP address
- Pods are ephemeral – what happens to IP traffic if they die?
- Pods in a service can have fixed IP addresses that stay the same even if the actual pod dies and is spun up again
- Services are k8s objects; they are created like any other

k8s Service Types

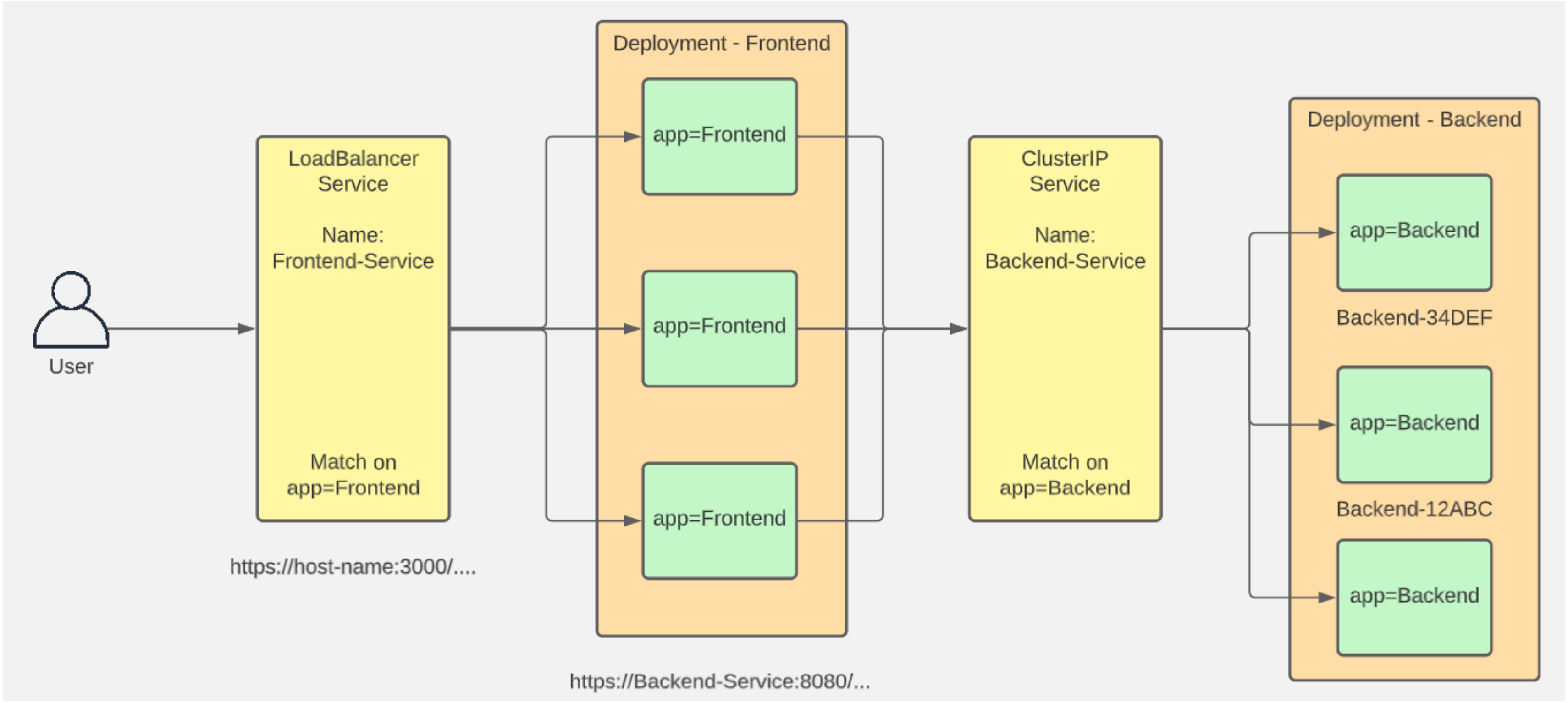
- ClusterIP (default): IP is internal to the cluster
- NodePort: IP for the port is exposed with a static IP address
- LoadBalancer: IP traffic to nodes is managed by external load balancer
- ExternalName: IP traffic is routed by DNS Name (foo.example.com)

Cluster Access – Internal vs. External

- You can access the cluster API using kubectl (“proxy”)
- You can get libraries for popular languages that can access the cluster
- k8s objects (pods) can also access the API internally
- This uses an internal DNS
- “kubernetes.default.svc” maps to the API server
- The pod usually uses a “sidecar” container to do this

Services

kubernetes



LAB:

Kubernetes – Services

Execute the lab available at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/k8s-services.

Container File System

- Each container in a pod has its own filesystem built by k8s
- Can be built from multiple sources, including:
  - Layers from the container image
  - Writable layer for the container
- Can be expanded with other sources like ConfigMaps and Secrets – mounted to a specific path in a container
- Volumes can provide another source of storage – volumes are defined and mounted to containers

EmptyDir

kubernetes

- Empty directory stored at pod level vs. container level
- Mounted as a volume into the container so visible there
- Any data stored there from the container remains in pod between restarts
- Replacement containers can access data from predecessors

HostPath

- Data physically stored on node in cluster
- Extends beyond lifecycle of pod
- Available to replacement pods (but only if run on same node)
- Mounted as a volume into the container like others
- However, can have issues:
    - ➢ In cluster with multiple nodes, limits usefulness
    - ➢ If not careful, can expose large blocks of host data

PersistentVolume

- Like pods (abstraction over computer) and services (abstraction over network), persistent volumes provide abstraction over storage
- k8s object that defines available section of storage
- Can be “mapped” to shared storage (like NFS) or locally (for dev/test purposes)

PersistentVolumeClaim

- Pods are not allowed to use persistent volumes directly
- Instead, pods claim using PersistentVolumeClaim object type in k8s
- Requests storage for a pod
- k8s handles matching up a claim to a persistent volume
- Provides virtual storage abstracted from actual storage

Dynamic Provisioning

- Static provisioning accomplished by explicitly creating persistent volume and then claim (k8s binds claim to volume)
- Dynamic provisioning supported as well
- You create the persistent volume claim and let persistent volume be created on demand by cluster
- Can leverage storage classes for defining and matching types of storage

Storage Classes

- Defined by three properties:
  - provisioner – component that creates persistent volumes on demand
  - reclaimPolicy – what to do with dynamically created volumes when claim is deleted
  - volumeBindingMode – eager vs. lazy binding/creation (at same time as claim creation or only when pod using the claim get created)
- Also, able to create custom storage classes defined by same properties

LAB:

Kubernetes – Persistent Volumes & Persistent Volume Claims

Execute the lab available at https://github.com/KernelGamut32/capital_group_edge_la_public/tree/main/labs/week02/k8s-pvs-and-pvcs.

Thank you!

If you have additional questions, please reach out to me at: erik-gross@pluralsight.com

DevelopIntelligence
A PLURALSIGHT COMPANY