Integration Testing with Service Containers

GUY SALTON

Guy Salton

Solutions Architect

codefresh

guy.salton@codefresh.io

Agenda

- What is integration testing?
- Introducing service containers
- Checking readiness of a service
- Preloading data to databases
- Launching a custom service
- Summary

https://github.com/codefreshdemo/cf-example-integration-tests

Unit vs Integration testing

Unit Test

In Unit test, small module or a piece of code of an application is tested.

Integration Test

In Integration test, individual modules combined together and as a group they are tested.

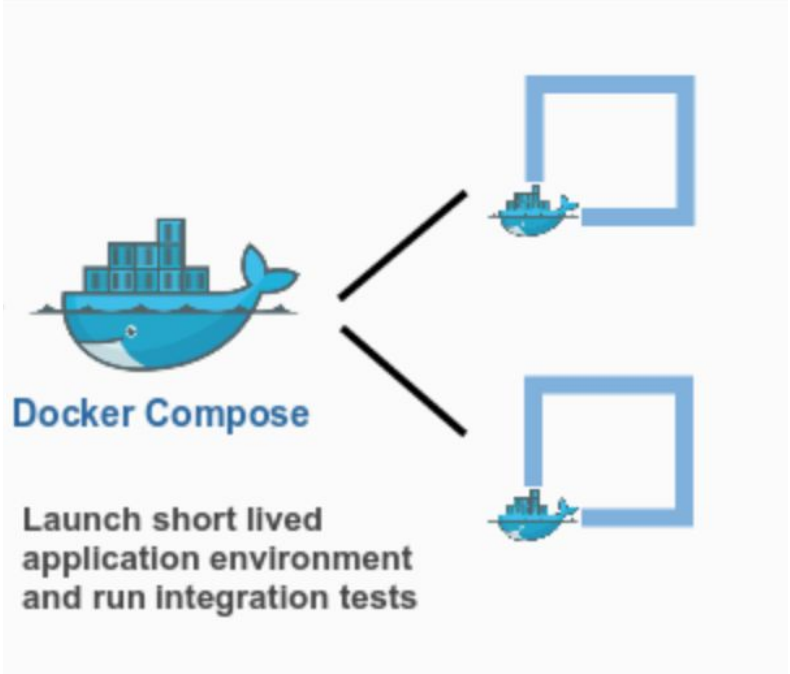
What is Integration testing?

- Testing the interface between multiple software units or modules
- The purpose of the integration testing is to expose faults in the interaction between integrated units.
- Usually need to launch either the application itself or one or more external services (such as a database) to run an integration test

Integration Tests with Docker Compose

Compose is a tool for defining and running multi-container Docker applications

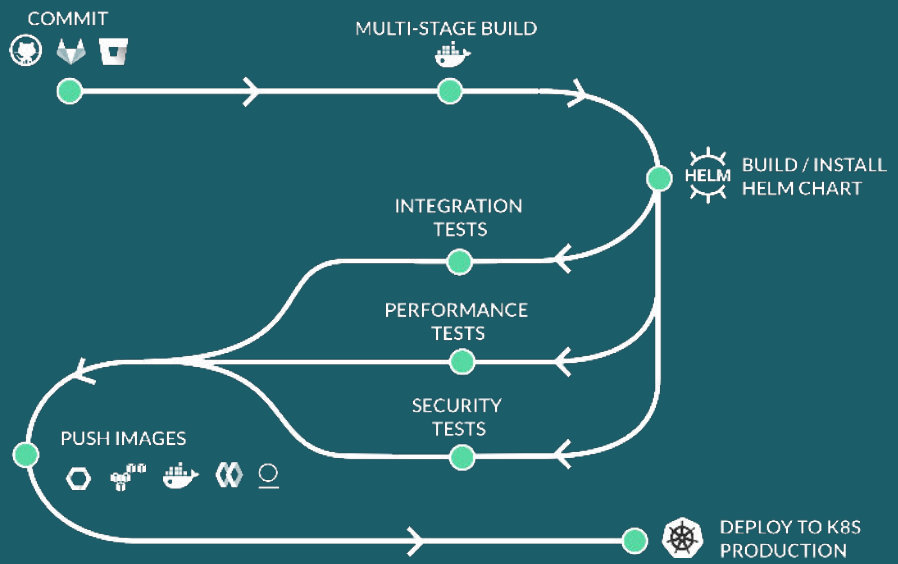
```yaml
web:
  build: .
  dockerfile: Dockerfile
  links:
    - redis
  ports:
    - "80:80"
redis:
  image: redis
```



Codefresh

The 1st container-native CI/CD Platform for Microservices

- **Container-native**
- **Intuitive & Robust**
- **Enterprise Ready**
- **Flexible Delivery**



Introducing Service Containers

- You can guarantee the order of service launch and their dependencies (a feature that is not even offered by vanilla docker-compose)
- You can use a special docker image to preload data to a database or otherwise initialize a service before tests are run
- The service containers can be attached on the whole pipeline instead of individual steps

Checking readiness of a service

```yaml
version: "1.0"
steps:
  main_clone:
    type: "git-clone"
    description: "Cloning main repository..."
    repo: "kostis-codefresh/my-back-end"
    revision: "master"
    git: github
  build_image:
    title: "Building Docker Image"
    type: "build"
    image_name: "my-backend-app"
    tag: latest
    dockerfile: "Dockerfile"
  run_integration_tests:
    title: Test backend
    image: 'my-front-end:latest'
    commands:
      # Backend is certainly up at this point.
      - npm run integration-test
    services:
      composition:
        my_backend_app:
          image: '${{build_image}}'
          ports:
            - 8080
      readiness:
        image: 'byrnedo/alpine-curl'
        timeoutSeconds: 30
        commands:
          - "curl my_backend_app:8080"
```

Checking readiness of a service

- **periodSeconds**: How often (in seconds) to perform the probe. Default to 10 seconds. Minimum value is 1.
- **timeoutSeconds**: Number of seconds after which the probe times out. Defaults to 10 seconds. Minimum value is 1.
- **successThreshold**: Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1. Must be 1 for readiness. Minimum value is 1.
- **failureThreshold**: failureThreshold times before giving up. In case of readiness probe the Pod will be marked Unready. Defaults to 3. Minimum value is 1

Demo 1:
NodeJS and MySQL

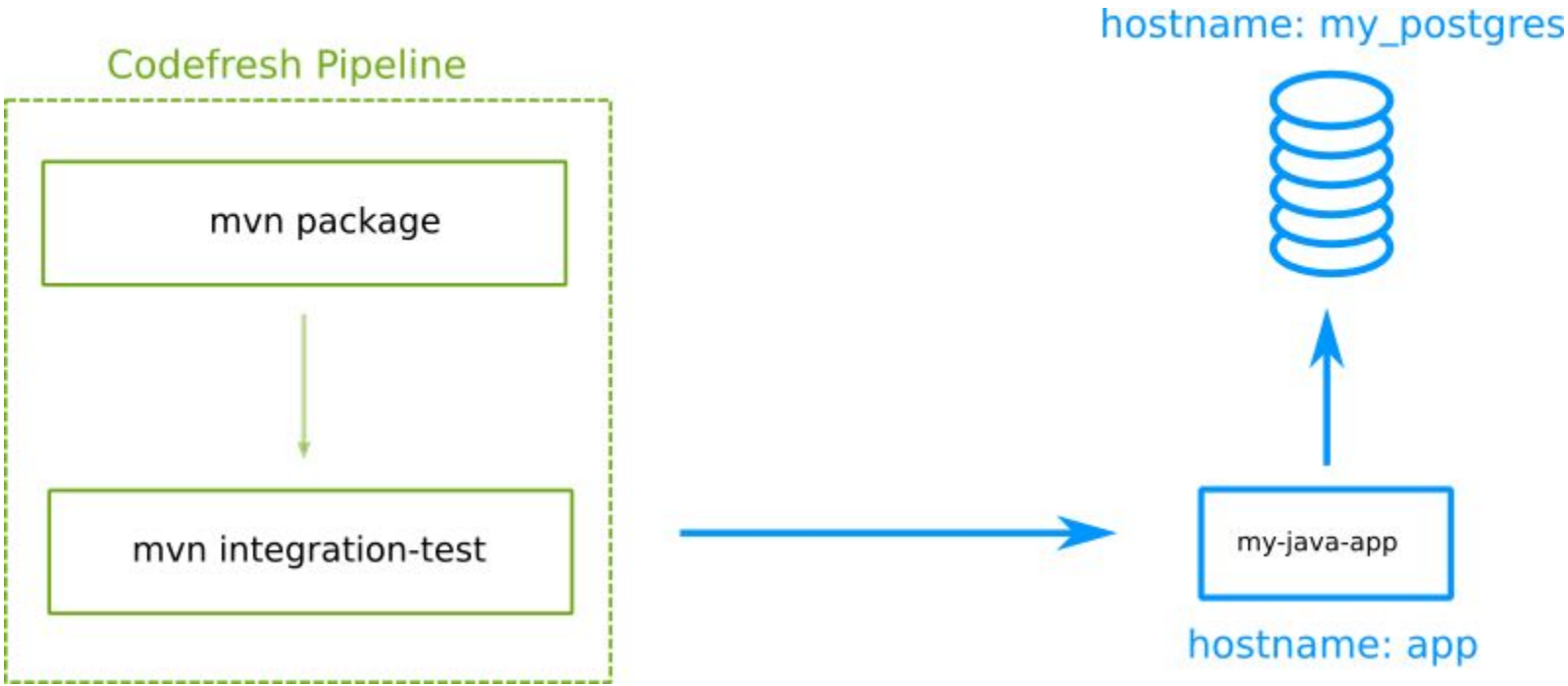
https://github.com/codefreshdemo/cf-example-unit-tests-with-composition

Preloading data to databases

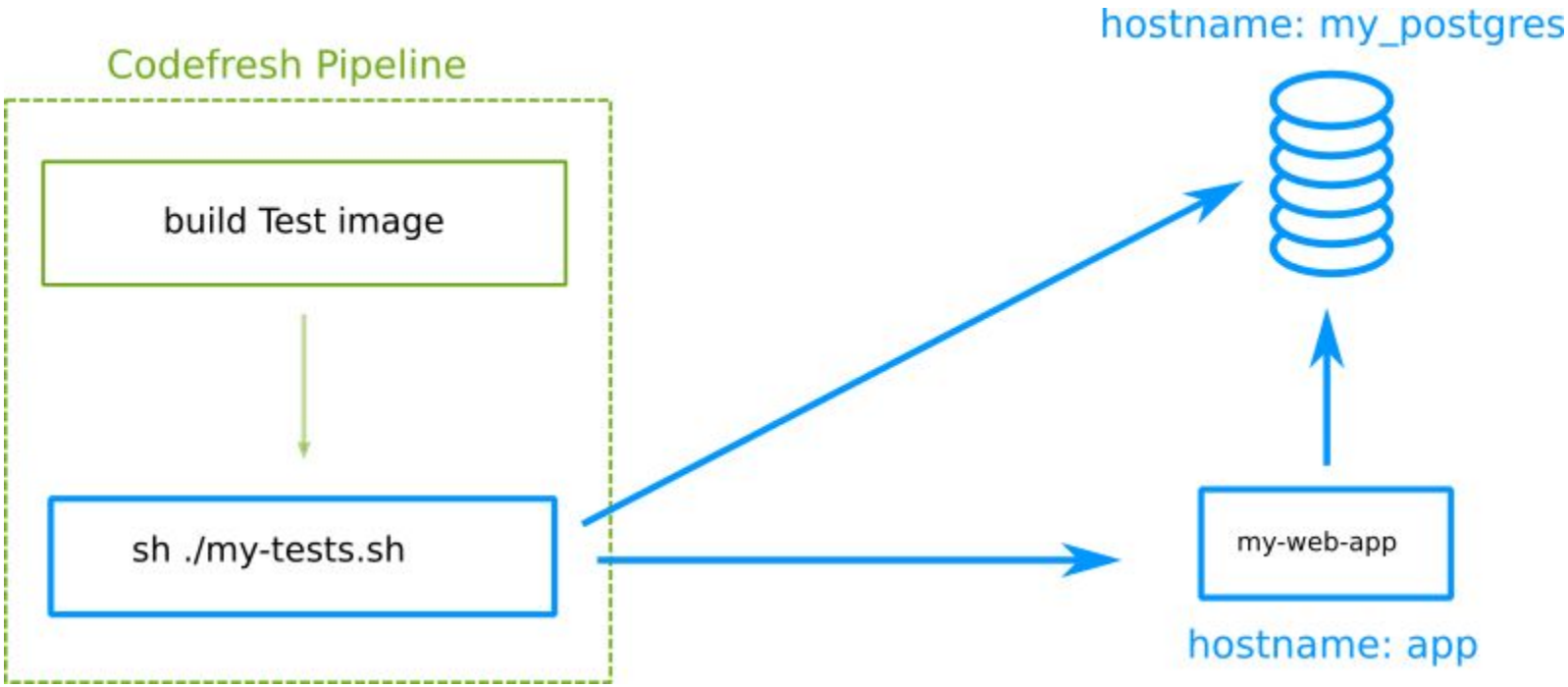
- A very common databases in integration tests is the need to preload some test data in the database.
- Sidecar services have a special setup block for this purpose.
- This way not only you can make sure that the database is up (using the readiness property explained in the previous section) but also that it is preloaded with the correct data.

Demo 2:
Rails and Postgres

Launching a custom service



Launching a custom service



Demo 3:
Launching a custom image

Integration Tests Best Practices

- Make sure that the hostnames used by your integration tests to access external services are not hard-coded
- Do NOT use localhost for an API endpoint (for MySQL service at hostname *my_db*, then your tests should use *my_db:3306* as a target)
- Even better - make the hostname completely configurable with an environment variable
- Make sure that your integration tests work fine with docker compose locally first

Summary

- Use Docker-Compose syntax in the pipeline to spin-up service containers
- Add the *readiness* block to guarantee the order of service launch and their dependencies
- Add the *setup* block to preload your database with the correct data for the integration test

Thank You!

Build Fast, Deploy Faster

Signup for a FREE account with UNLIMITED builds

& schedule a 1:1 with our experts at https://codefresh.io

guy.salton@codefresh.io