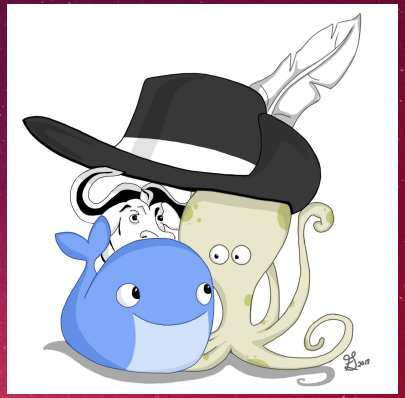
CONTINO

Improving the Developer Experience with 3 Musketeers



Welcome our newest developer, Maxine Chambers, to the Unicorn Project at Parts Unlimited!

Day 0

- Account created with default entitlements
- Manager hopefully adds you to a few groups that he remembers you will need
- Senior engineer notified that you are joining tomorrow and is your buddy. Calendar already overbooked.

Day 1

- After HR onboarding, you meet your manager and your buddy. They introduce you to the rest of the team.
- They walk you through an overview of the application
    - It's a modern microservices architecture that leverages "the Cloud" and Kubernetes.
- After lunch, your credentials are ready.
- Your first task is to build the application locally.
- You're given a link to Confluence and a link to GitHub.
- Also, if you could update any out of date documentation, that would be really helpful for the next person
- The Confluence page hasn't been updated in over a year…and start reading

Days 1, 2, 3, ...

- Step 1
  - Download JDK (version 8!)
  - Download Gradle (no Gradle Wrapper)
  - Install Postgres (version 10) & run some scripts to copy a development database locally
  - Clone Git repo (README is empty so you're going off the Confluence Page
- Step 2
  - Run '`gradle build`'
    - Build fails due to wrong JDK (they upgraded a few months ago and forgot to update the onboarding documentation)
  - Download JDK version 11
  - Run '`gradle build`'
    - Build fails due to authentication errors (blank password)
    - Ask a teammate what the password needs to be
  - ...repeat

Day 4 (It builds locally!)

- Make your first commit on a new feature
- Build passes locally
- You want to run the application
  - `'gradle bootRun'`
- It works!
- Push to feature branch
- Jenkins build fails
  - SonarQube analysis failed (not executed via Gradle)
  - Address issues, commit, push, … repeat…
- Jenkins build passes
- Deployment fails
  - Application deployed as a Docker container with a Helm chart (also not executed via Gradle)

And so it goes...

Andy Ochsner

Technical Principal @ Contino

Things that spark joy

- Agile/Lean Delivery
- DevOps
- Software Architecture
- Software Development
- CI/CD
- TDD/BDD

https://twitter.com/aochsner
https://www.linkedin.com/in/aochsner

About Contino

We work with the World's Leading Brands to help them with measurable transformation, through the adoption of Enterprise DevOps, Cloud Native Computing and Data Platforms.

We help our clients build their own innovation engine which allows them deliver better software faster and more efficiently as part of their digital transformation ambitions.

360+
People

The deepest pool of DevOps, data & cloud transformation talent in the industry

5
Global offices

We can scale rapidly to support diverse client requirements across the globe

300+
Engagements

More DevOps transformation executed than any other professional services firm

150+
Customers

Specializing in helping the world's leading brands accelerate digital transformation

JPMorgan Chase & Co. Nasdaq TD Ameritrade verizon AMERICAN EXPRESS Morgan Stanley

Agenda

01 | 3 Musketeers who & why

02 | Docker / Compose / Make

03 | Demo

04 | Lessons Learned & Gotchas

05 | Alternatives

06 | Q&A/Discussion

Let’s meet the 3 Musketeers!

Docker

- Builds, tests, and deployments can happen independent of the host system.
- Jenkins VS local execution.

Compose

- Orchestrating multiple docker containers.
- Simplifies makefile docker commands.

Make

- Multi-platform build automation tool.
- Uses Makefiles and associated targets.

3 Musketeers is good for your health

Consistency

Run the same commands no matter where you are: Linux, MacOS, Windows, CI/CD tools that supports Docker like Travis CI, CircleCI, and GitLab CI.

- 1st Day Productivity
- 1 type of Build Agent ever
- Savings of $4000-$7000 per new joiner
- All large enterprises need this, especially those with outsourcing and high turnover squads

Control

Take control of languages, versions, and tools you need, and version source control your pipelines with your preferred VCS like GitHub and GitLab.

- New level of smart endpoint dumb pipeline.
- Platform team become enabler, not version police
- No need for bloated CD tools with 3000+ integrations and plugins out of box

Confidence

Test your code and pipelines locally before your CI/CD tool runs it. Feel confident that if it works locally, it will work in your CI/CD server.

- No more “but the tests are passing on my laptop!”
- Run full web UX and complex db integration tests all within containers
- Change CD tool is not a problem, your app code is self sufficient

Docker

Docker

- Docker is the most important musketeer of the three.
- Many tasks such as testing, building, running, and deploying can all be done inside a lightweight Docker container.
- The portability of Docker ensures you can execute the same tasks, the same way, on different environments like MacOS, Linux, Windows, and CI/CD tools.

Docker Example

```
$ docker run -rm alpine echo ‘Hello world!’
```

Docker Compose

Docker Compose

- Docker Compose, or simply Compose, manages Docker containers in a very neat way.
- It allows multiple Docker commands to be written as a single one, which allows our Makefile to be a lot cleaner and easier to maintain. Volume mounts, working dir, environment variables, port mapping, etc.
- Testing also often involves container dependencies, such as a database, which is an area where Compose really shines. No need to create the database container and link it to your application code container manually — Compo takes care of this for you.

Docker Compose Example

```
# docker-compose.yml
version: '3'
services:
  alpine:
    image: alpine


$ docker-compose run --rm alpine echo ‘Hello World!’
```

Make

Make

- Make is a cross-platform build tool to test and build software and it is used as an interface between the CI/CD server and the application code.
- Originally created Stuart Feldman at Bell Labs in 1976!
- A single Makefile per application defines and encapsulates all the steps for testing, building, and running that application.
- Of course other tools like rake or ant can be used to achieve the same goal, but having Makefile pre-installed in many OS distributions makes it a convenient choice.

Make example

```
# Makefile

# echo calls Compose to run the command
# "echo 'Hello, World!'" in a Docker container
echo:
    docker-compose run --rm alpine echo 'Hello, World!'


$ make echo
```


Demos

Lessons Learned & Gotchas

Make

- `make` is a powerful tool that can do a lot of things. However, the intent is to use `make` as a simple interface between your build tool and your build steps.
- Learn the basics: Tabs not spaces, .PHONY, := variable assignment
- As you should only be using the most basic features of `make` it shouldn't be difficult to implement.
- Having a clean `Makefile` is key.
- It helps to understand it quickly and is easier to maintain.
- If your `Makefile` is getting complex, consider if all the logic should be in `make` or if you can simplify it or push it down the stack (into Compose).

Environment Variables

- `Make` is shell-like, but it's not a shell. You can't set variables to be used on future `make` calls, other than through a file.
- Variables in `make` can come from the environment in which `make` is run.
- Every environment variable that `make` sees when it starts up is transformed into a `make` variable with the same name and value.
- However, an explicit assignment in the `Makefile`, or with a command argument, overrides the environment.
- By default, only variables that came from the environment or the command line are passed to recursive invocations. You can use the `export` directive to pass other variables.
- Docker/Compose can leverage an env_file (typically `.env`) which is not committed to Git. Put in `.gitignore`

Gotchas

- Don’t use :latest as it won’t be idempotent
- Make sure you clean up after yourself, especially on shared build agents when you build and run containers
- File permissions
    - By default Docker runs as root, therefore files/folders created on volume mounts are owned by root
    - Creating a nonroot user on image build solves some of that, however filesystem permissions are still based on user ids which may not be consistent from host to host and user to user
- Secret management is still challenging. Use the same solution across all of your environments
- Ensure you’re using an up to date version of Docker & Docker Compose
    - Older versions of Docker Compose didn’t support .env files to pass in environment variables from the host
    - Can implement make targets to download/install on the fly

Alternatives

Choose your own musketeers

Who says make, compose, docker are the perfect combo?

The key to Consistancy, Control and Confidence is not only in the specific combination of make, compose, docker but instead in the agreement of your team and your organization to use a standard that's easy for everyone to follow.

Consider:

- Do most of your development teams have easy access to make, compose, docker - or can you agree/reach a state where this is possible?
- Do you want to replace make with something else? (gradle, npm, shell) Sure thing, just agree as a team what would be more easily to install and maintain
- Do you want to start with make and docker and introduce compose only when the complexity demands it?

https://3musketeers.io/docs/patterns.html

2 Musketeers

- Docker & Bash
- https://2musketeers.sh/

The 2musketeers.sh script detects whether the current script is running in a Docker container, and, if not, builds a container to wrap the current script, setting up the necessary volume mappings, passing in the current environment variables, mounting the current directory, and creating the necessary uid & gid mappings, and finally (but quickly) runs itself.

The 2musketeers.sh is a pattern that is designed to be checked into source along with your code, and modified to suite your particular test, build, run and deploy needs.

Bootstrap scripts

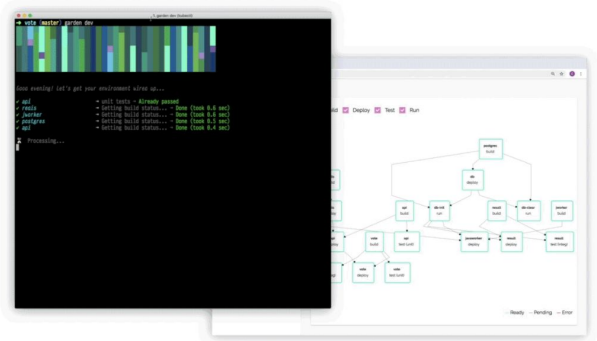
- Bash & Homebrew (or Powershell & Chocolatey)
- Run a script, have a standard development layout
- Doesn’t address CI parity
- Hard to make cross platform

Garden

Garden is a developer tool that automates your workflows and makes developing and testing Kubernetes applications faster and easier than ever.

- Keep your development environment **up-to-date as you code**, and get rapid feedback.
- **Develop and iterate as quickly with remote clusters as you do locally**, and share development clusters with your team. With remote clusters you can even **run Garden without Kubernetes or Docker installed** on your machine!
- **Start simple** and grow complexity gradually as your needs evolve. Use simplified abstractions to start, and move to raw Kubernetes YAML, Helm, etc. only when you have to.
- **Simplify your CI** by running the same commands and configuration during development and testing—and use the same build and test caches!

> *If you're using Garden or if you like the project, please ★ star this repository to show your support 💖*




Q & A

Links

- Demo Code - https://github.com/aochsner/omaha-devops-3musketeers
- 3M Site - https://3musketeers.io
- Make Docs - https://www.gnu.org/software/make/manual/make.html
- Docker Docs - https://docs.docker.com
- Docker-Compose Docs - https://docs.docker.com/compose/
- Katacoda - https://www.katacoda.com/contino/courses/3musketeers
- https://medium.com/@drew.khoury/optimizing-for-dx-the-developer-experience-f37fe168642d
- https://github.com/drewkhoury/3musketeers-master-class
- https://github.com/contino/docker-terraform
- https://amaysim.engineering/the-3-musketeers-how-make-docker-and-compose-enable-us-to-release-many-times-a-day-e92ca816ef17
- https://alexharv074.github.io/2019/11/07/when-3-musketeers-are-two-too-many.html

CONTINO

Thank You

London
london@contino.io

New York
newyork@contino.io

Melbourne
melbourne@contino.io

Sydney
sydney@contino.io

Atlanta
atlanta@contino.io

www **contino.io** **continohq** **contino**