



INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS

CONTAINERS

STUDENT GUIDE

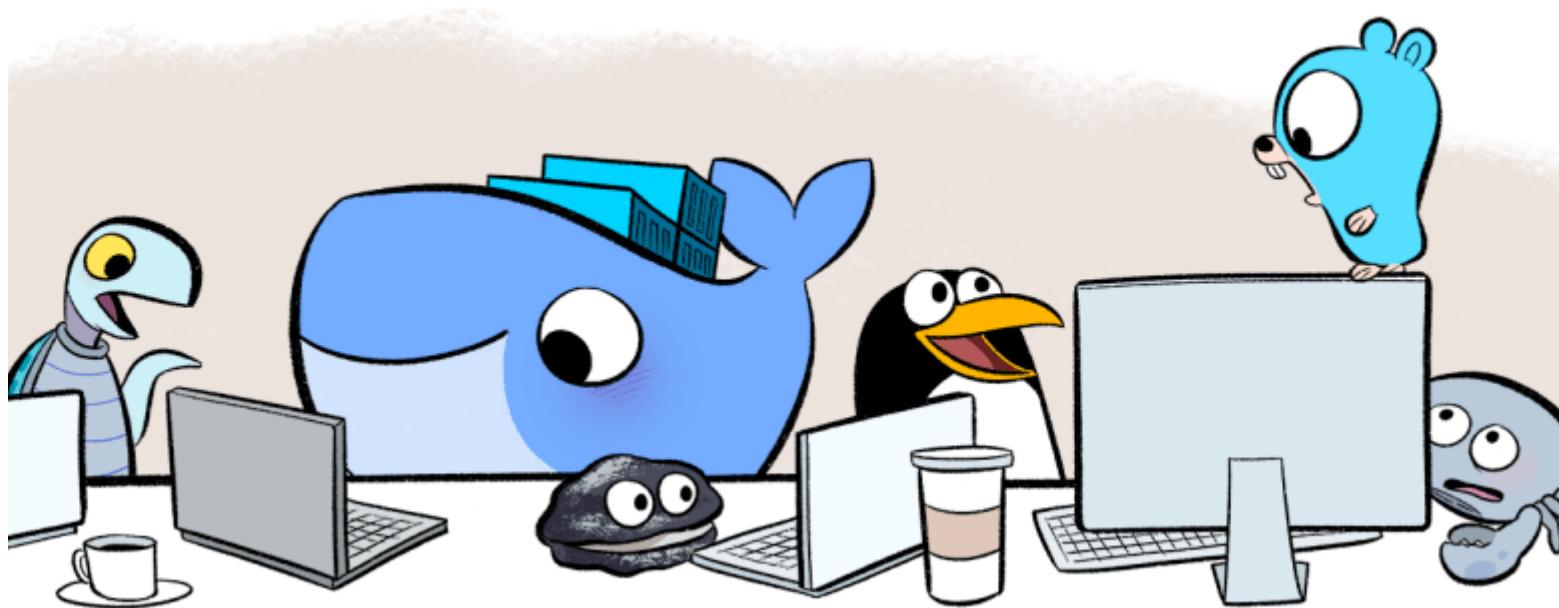
From Docker to Kubernetes

TABLE OF CONTENTS

Containers Basics
Installing Docker
Running OCI images in Docker
Building OCI Images
Application Architecture
Application Development
Publishing OCI Images
Ports & Hosts
Docker Networks
Docker Volumes
Docker-Compose
Develop With Docker-Compose
Service Discovery
Observability
Application Monitoring
Production Challenges
Kubernetes Basics
Kubernetes UIs
Kubernetes CLI
Kubernetes API
Pods
Controllers
Services
Ingresses
Data Management
ConfigMaps & Secrets
Operators
Deploy Workflows
Second Day Operations
Course Conclusion

Course Objective

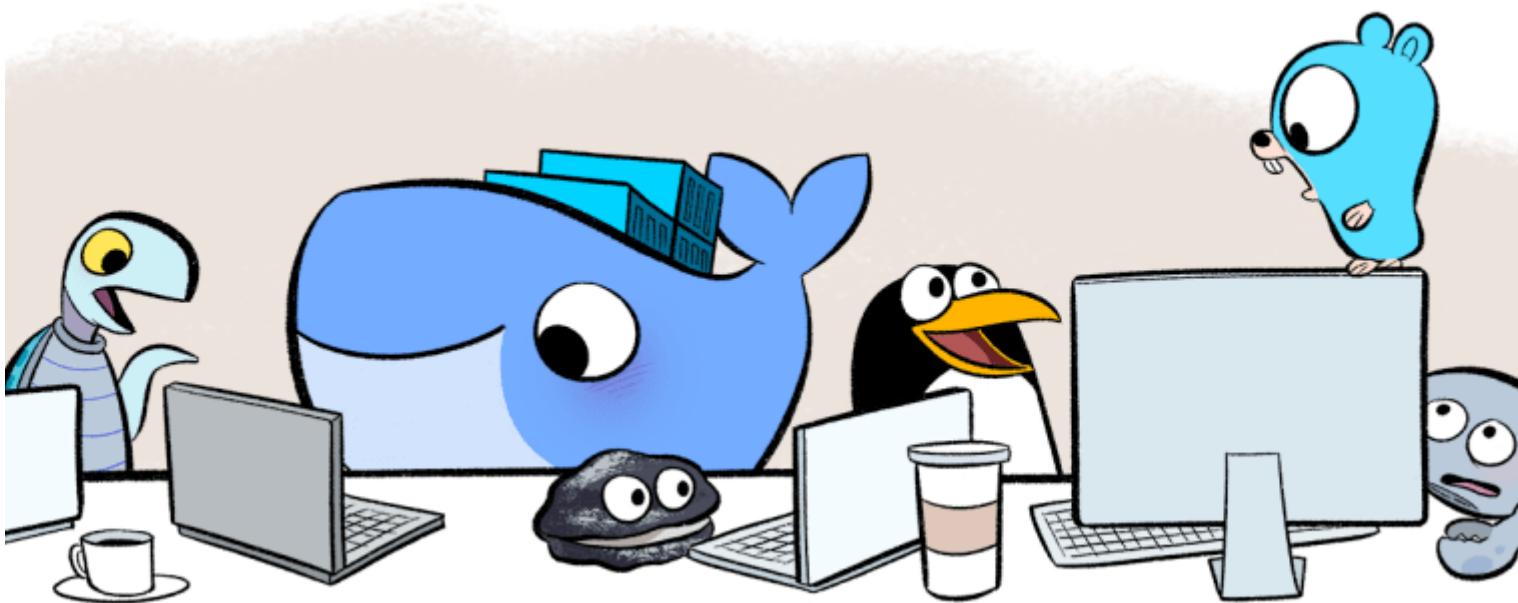
After completing this course, you will be able to use Docker to build and run images, as well as orchestrate Docker containers using Docker Compose and Kubernetes, and prepare a stack for production.



Course Overview

You will:

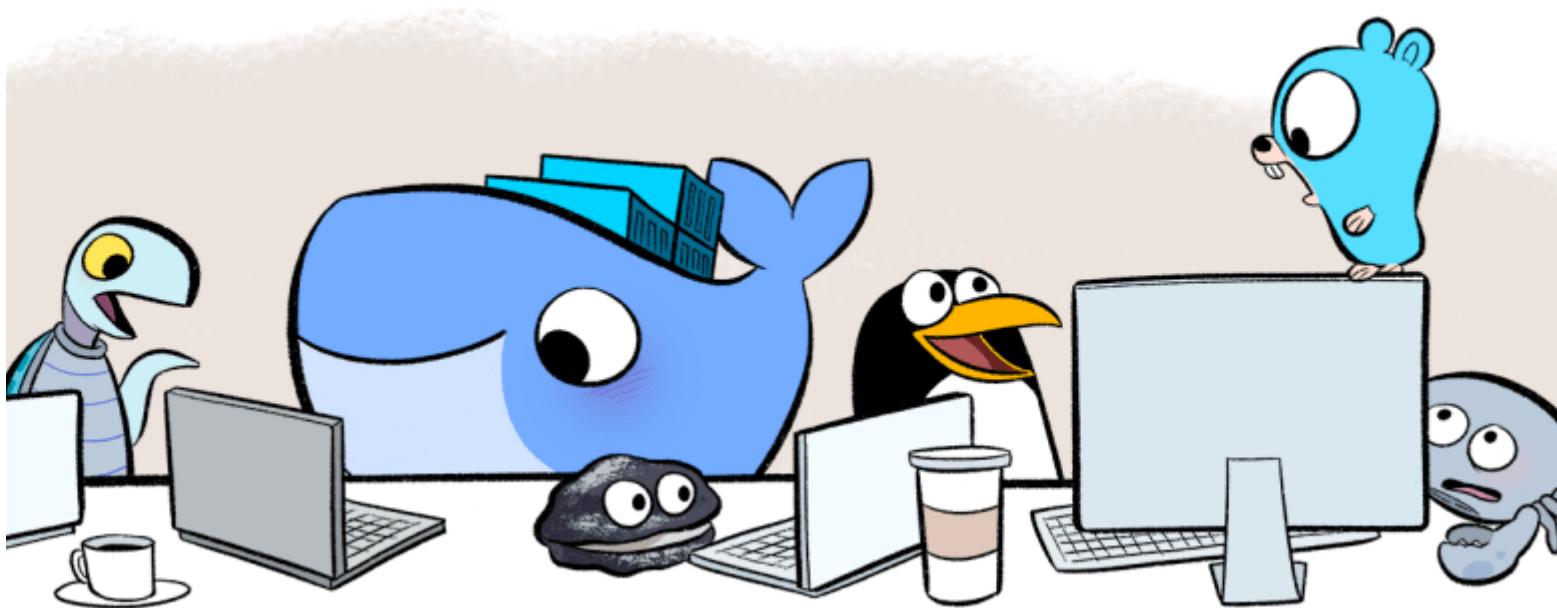
- Use the Docker CLI to run & build containers
- Use the `docker-compose.yml` format to orchestrate containers into a stack
- Make containers share volumes and communicate with each other
- Learn about Kubernetes history and concepts
- Use the `kubectl` command to create, update, destroy Kubernetes Objects
- Learn about the Kubernetes API
- Learn about the various Kubernetes Objects



Course Agenda: Day 1

Day 1: Docker Basics

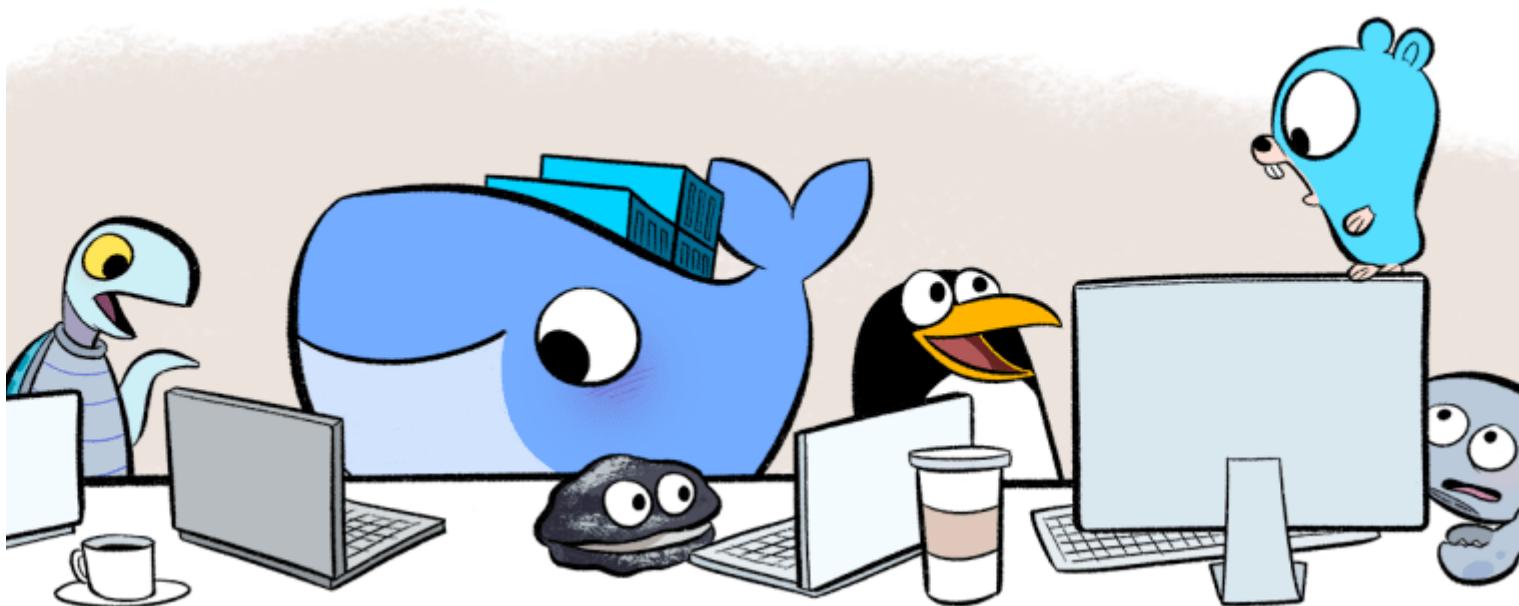
- Container basics;
- Installing Docker;
- Running containers;
- Building container images using a Dockerfile;
- Publishing images to a registry.



Course Agenda: Day 2

Day 2: Container Orchestration

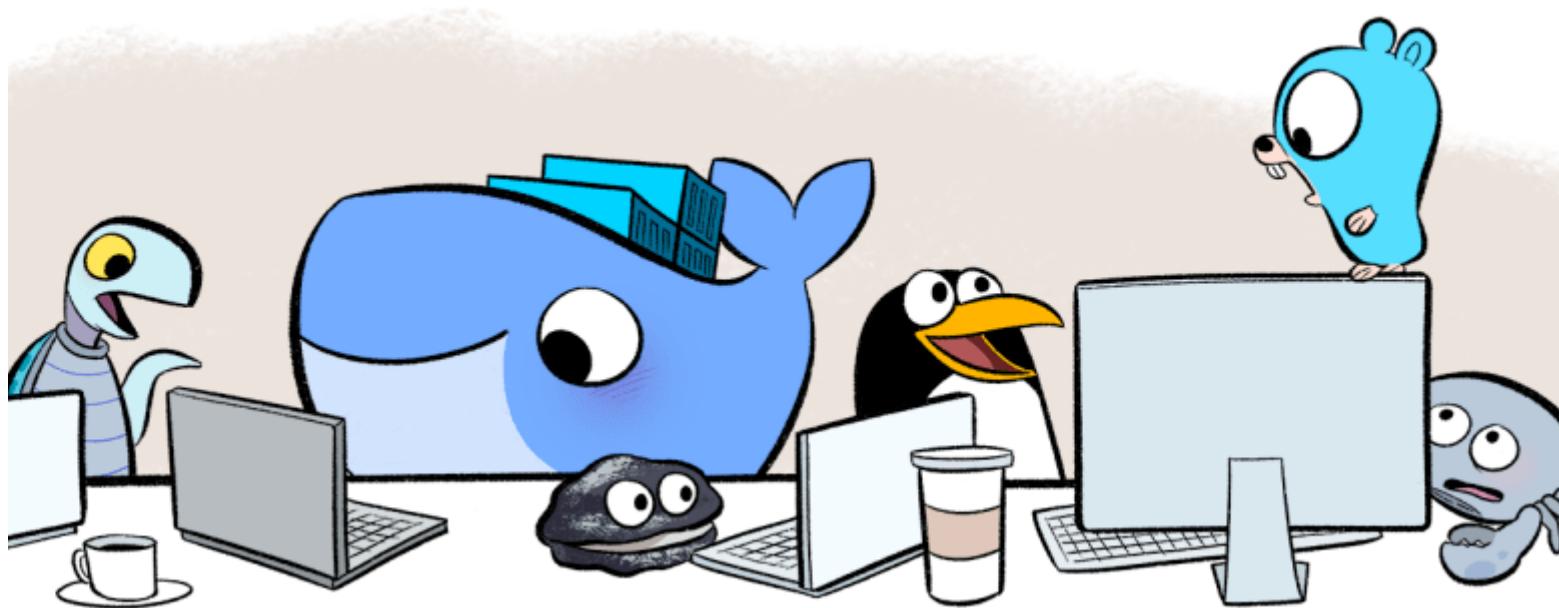
- Ports & Hosts;
- Volumes;
- Using docker-compose;
- Service Discovery;
- Production Challenges.



Course Agenda: Day 3

Day 3: Kubernetes Basics

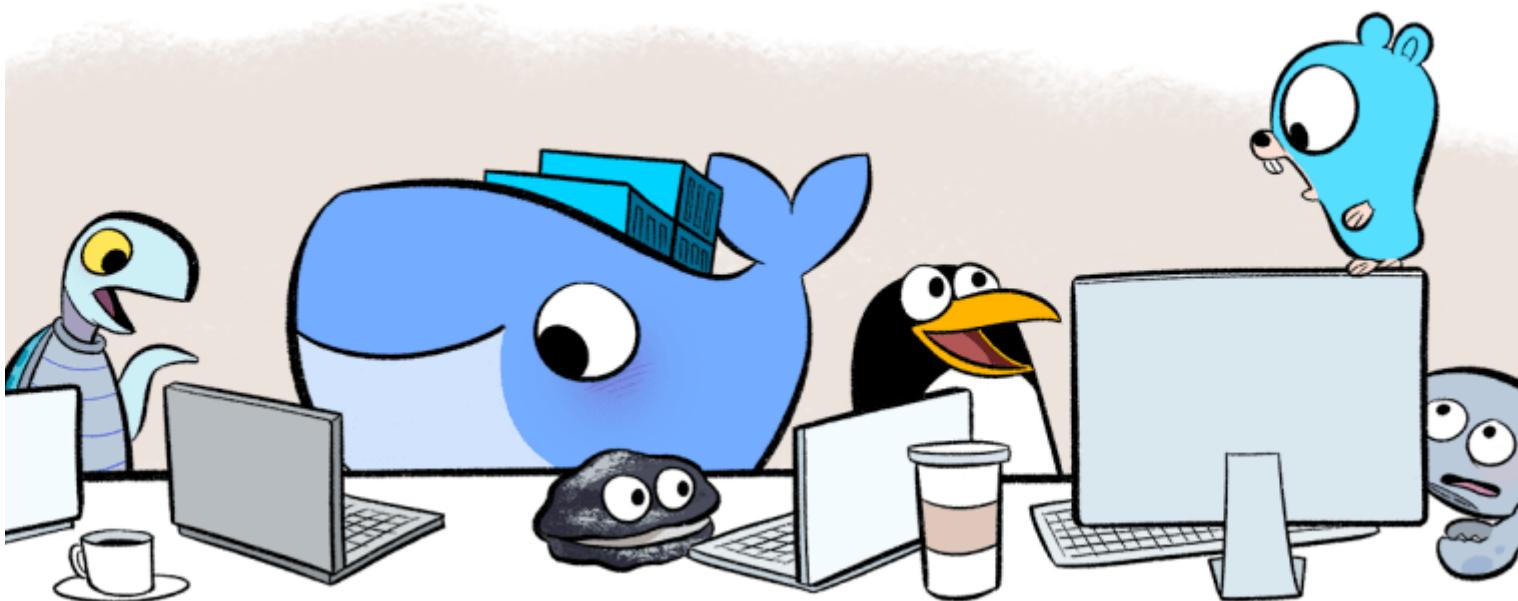
- Orchestration and Kubernetes Basics;
- Web UIs (dashboard, Rancher, OpenShift);
- CLI (kubectl);
- API and YAML objects.



Course Agenda: Day 4

Day 4: Kubernetes Resources

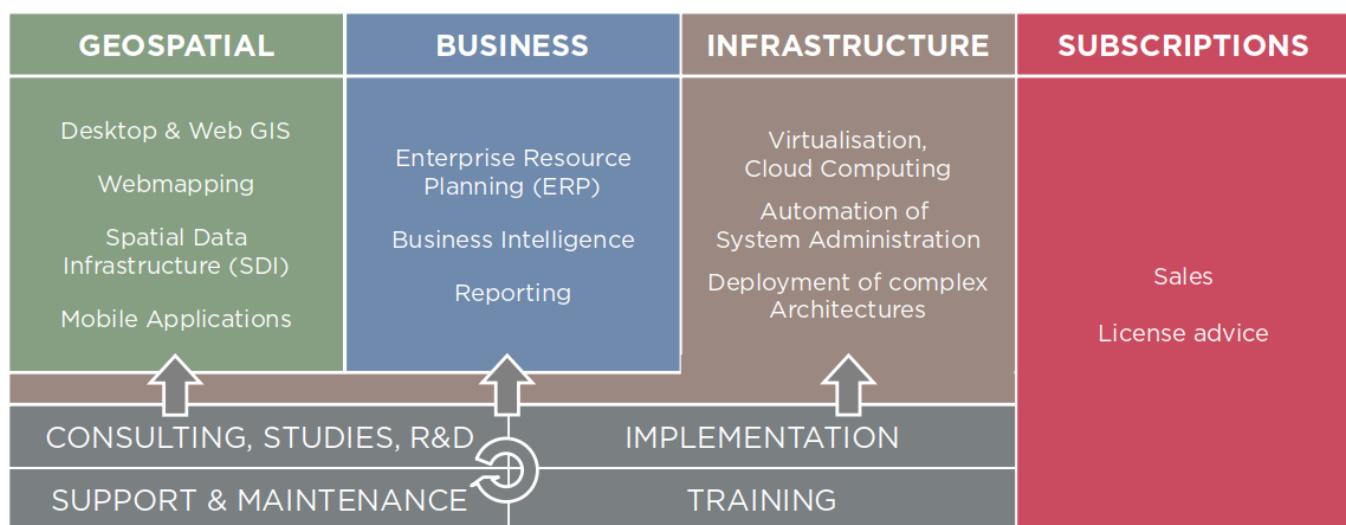
- Pods;
- Controllers;
- Services;
- Ingress;
- Data Management and Volumes;
- Config Maps & Secrets;
- Packaging (Helm & ArgoCD).



About Camptocamp

Open Source specialist, innovative company in the software implementation of:

- Geographic Information Systems (**GIS**)
- Business Management (**ERP**)
- Server Management (**IT Automation and Orchestration**)
- Subscriptions



About Camptocamp: locations

Present in three countries:

- Switzerland (Camptocamp SA)
- France (Camptocamp France SAS)
- Germany (Camptocamp GmbH)



About Camptocamp: Infrastructure department

- Involved in open-source communities (Puppet, Terraform, Rancher)
- **23** Systems Administrators
- Manages ~**1000** servers
- Partnerships:



Red Hat Premier Partner



Amazon AWS
Consulting Partner



Puppet Partner &
Authorized Training Partner



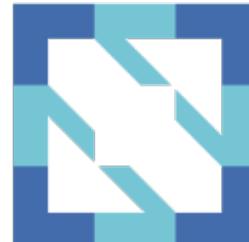
A1 Digital Partner



Kubernetes
Certified Service Provider



Kubernetes
Training Partner



Cloud Native
Computing Foundation
Silver Member



Linux Foundation
Silver Member

CONTAINERS BASICS

Lesson 1: Containers Basics

Objectives

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

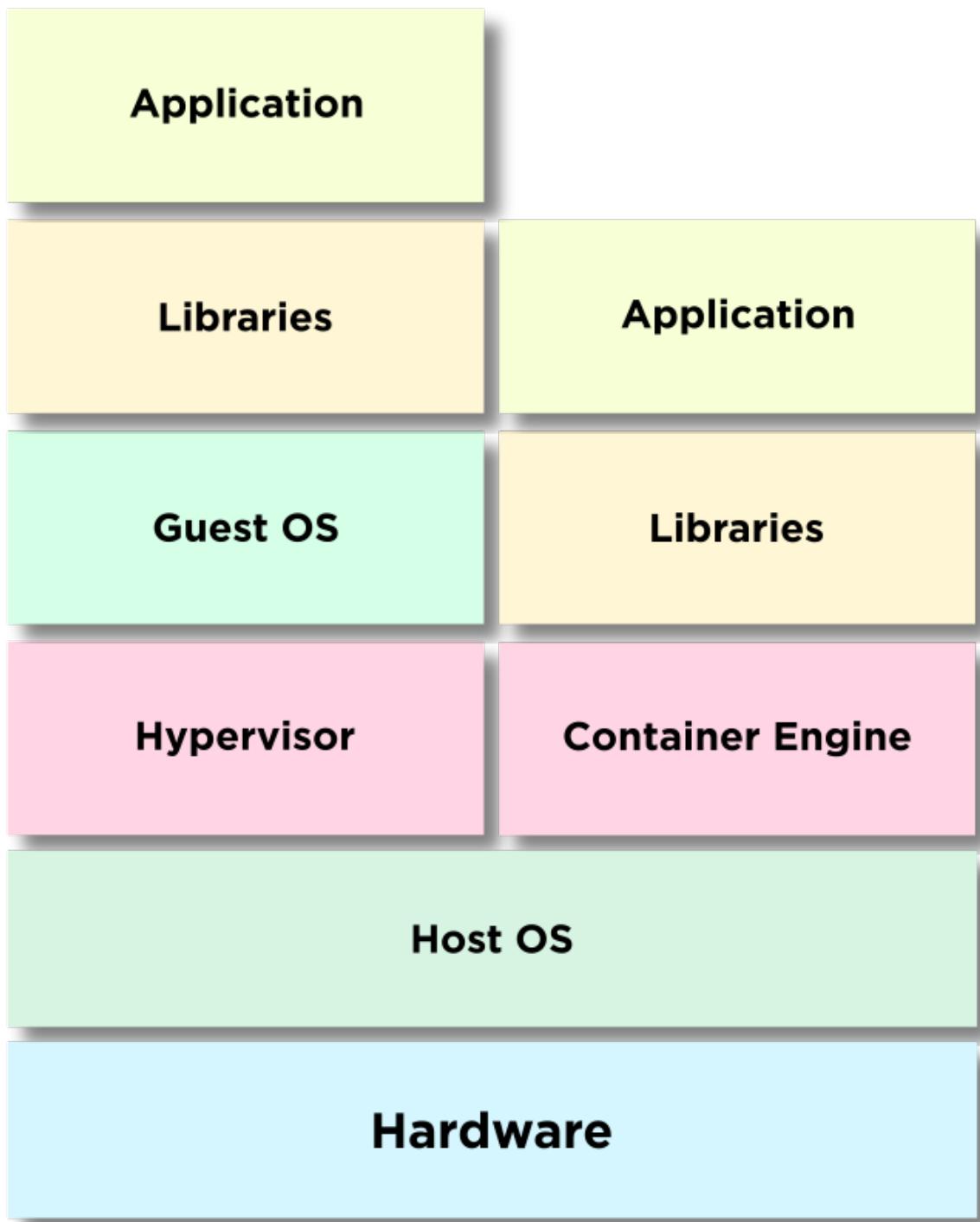
- understand the difference between virtualization and containers
- know the various isolation namespaces in cgroups

OS-level Virtualization



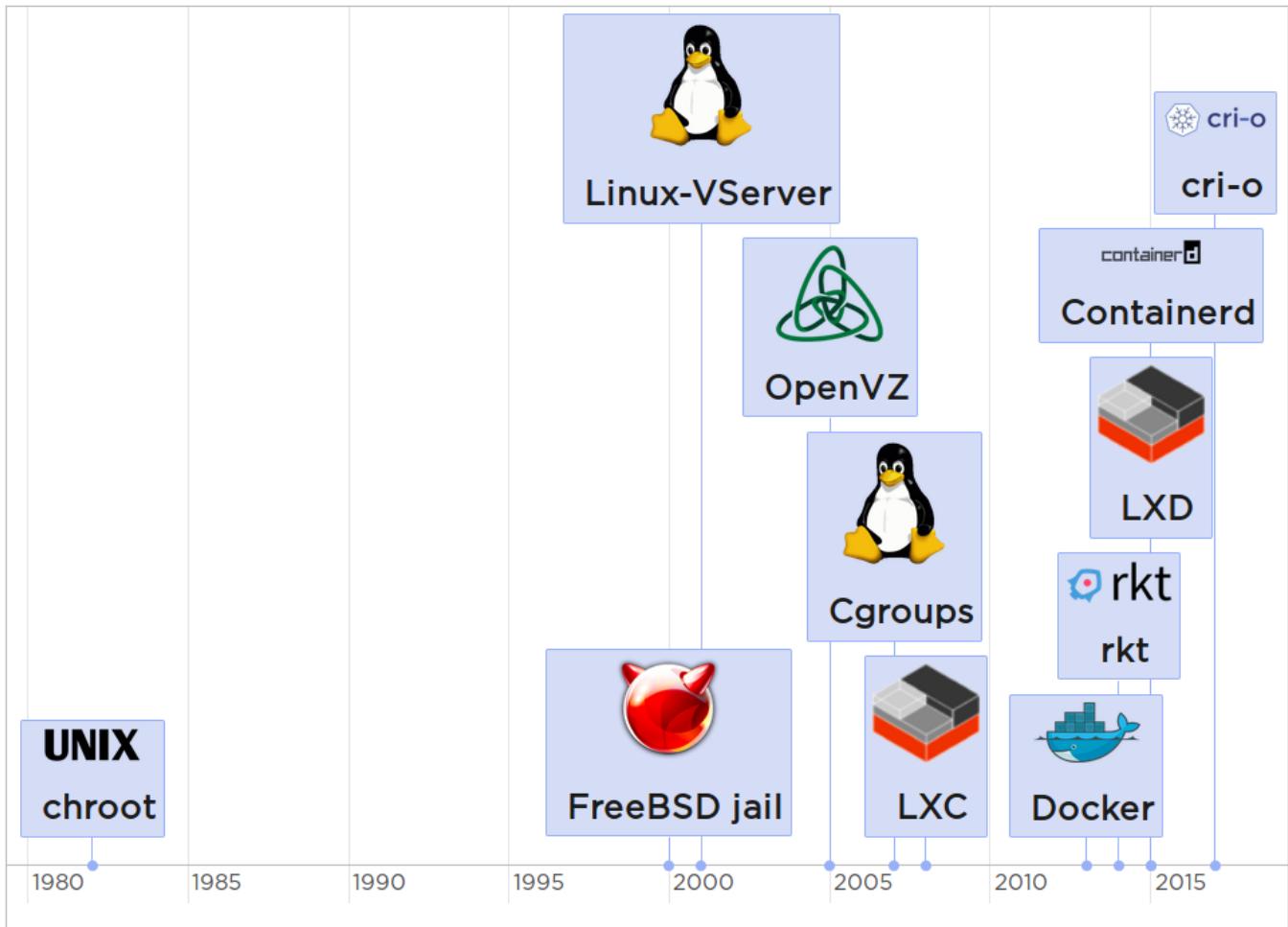
Operating-system-level virtualization is a server-virtualization method where the kernel of an operating system allows for multiple isolated user-space instances, instead of just one. Such instances (sometimes called containers, software containers, virtualization engines (VE), virtual private servers (VPS), or jails) may look and feel like a real server from the point of view of its owners and users. On Unix-like operating systems, one can see this technology as an advanced implementation of the standard chroot mechanism. In addition to isolation mechanisms, the kernel often provides resource-management features to limit the impact of one container's activities on other containers. ([Wikipedia > Operating-system-level_virtualization](#))

Virtualization vs Containers



Containers History

Aka Operating-system-level virtualization



Chroot

- Basic file system isolation
- No CPU isolation
- No network isolation
- No root privilege isolation

Useful to "jail" the file system accessible to a process.



```
$ sudo chroot /var/chroot/mychroot
```

Cgroups



cgroups (abbreviated from control groups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes. [Wikipedia > Cgroups](#)

Namespace isolation:

- **PID**: processes
- **Network**: network interfaces, firewall rules, routing tables, etc.
- **UTS**: hostname
- **Mount**: file system
- **IPC**: inter-process communication
- **User**: user IDs

Unshare

Run program with some Cgroups namespaces unshared from parent

```
$ sudo unshare --fork --pid --mount-proc -- readlink /proc/self
$ unshare --map-root-user --user          -- sh -c whoami
$ sudo unshare --uts                   -- sh -c 'hostname thing && hostname'
$ unshare --user                      -- ls -la
$ sudo unshare --ipc                  -- ipcs
```

Notes:

See <https://gist.github.com/jfrazelle/c771c63c376236bd419c> for more

Checkpoint: Containers Basics

- Containers
 - have their own libraries
 - have a separate network namespace
 - run their own kernel
 - embark a full Operating System

- Which virtualization engines are based on Cgroups?
 - Chroot
 - FreeBSD jail
 - OpenVZ
 - LXC
 - Docker

INSTALLING DOCKER

Lesson 2: Installing Docker

Objectives

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

- install Docker on your machine

Installing on Ubuntu

- Uninstall previous versions:

```
$ sudo apt-get remove docker docker-engine
```

- Install docker repository and Docker's official GPG key, and setup repository:

```
$ sudo apt-get install apt-transport-https \
  ca-certificates curl software-properties-common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] \
  https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

- Install Docker

```
$ sudo apt-get update
$ sudo apt-get install docker-ce
```

- Test installation

```
$ docker run hello-world
```

Notes:

- Adapt `ubuntu-xenial` to your OS (e.g. `debian-jessie`)

Installing on Mac OS X and Windows

- Docker is a Linux based technology
- the Windows and Mac OS X version uses virtual machines to provider Docker servers

Lab 2.1: install Docker



Objective

- Install Docker on your machine

Steps

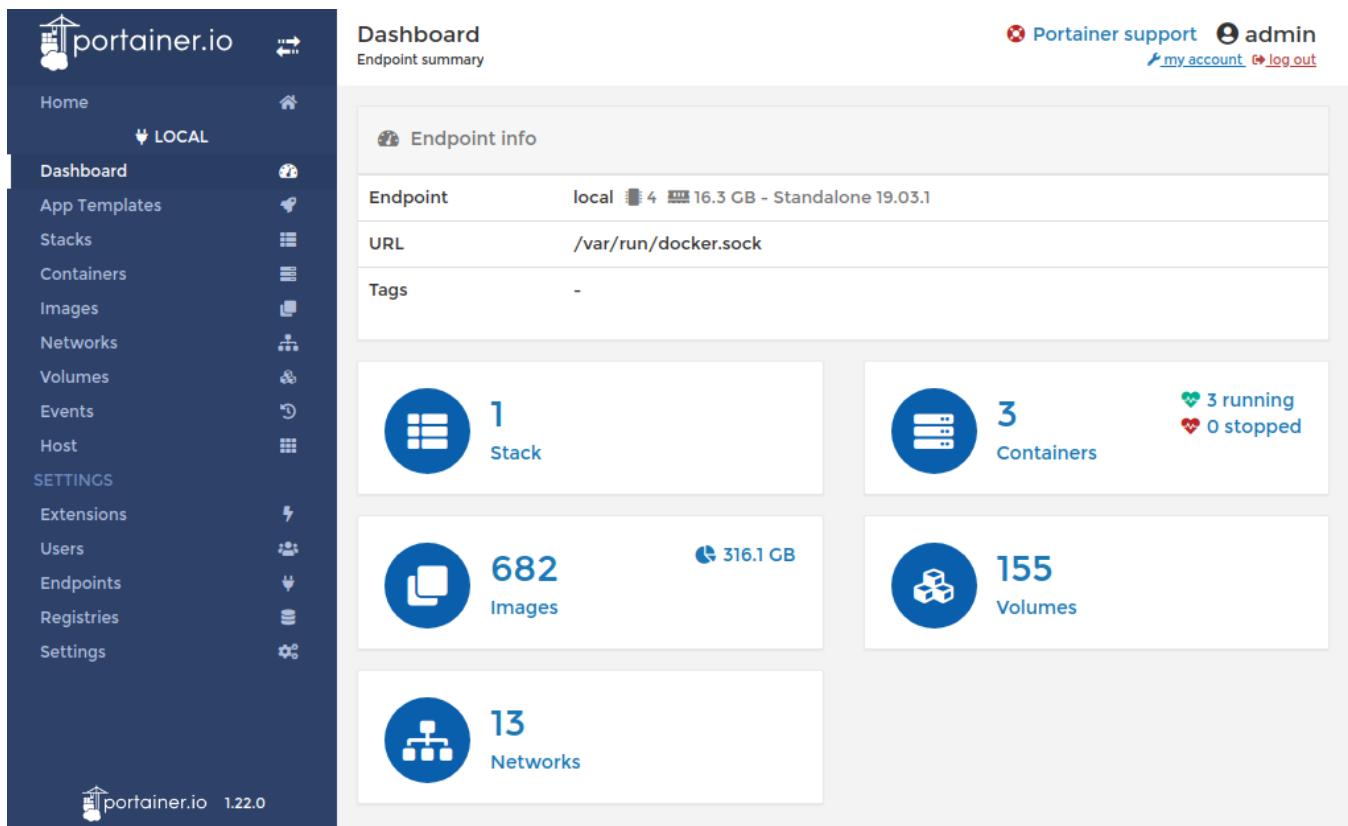
- Download/install docker-engine on your machine
- Give your user the necessary permissions to run Docker
- Run `docker version`, `docker info` and `docker run hello-world` to verify the installation

Questions

- The docker package provides
 - the dockerd daemon
 - the docker client
 - Kubernetes

Portainer

- Simple Web-based Docker UI
- Supports local or remote socket



The screenshot shows the Portainer dashboard interface. On the left is a sidebar with navigation links: Home, Dashboard, App Templates, Stacks, Containers, Images, Networks, Volumes, Events, Host, SETTINGS, Extensions, Users, Endpoints, Registries, and Settings. The main content area is titled "Dashboard" and "Endpoint summary". It displays the following information:

- Endpoint info:** local (4 cores, 16.3 GB - Standalone 19.03.1), URL: /var/run/docker.sock, Tags: -
- Stacks:** 1 Stack
- Containers:** 3 Containers (3 running, 0 stopped)
- Images:** 682 Images (316.1 GB)
- Volumes:** 155 Volumes
- Networks:** 13 Networks

Lab 2.2: install Portainer



Objective

- Launch a Portainer container

Steps

- Run `docker run -d -p 8000:8000 -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer`
- Access <http://localhost:9000> on your machine

Questions

- Can Portainer display containers from all classroom nodes?
 - Yes
 - No
- Can Portainer access Docker information from another node?
 - Yes
 - No

Checkpoint: Installing

- A docker client can control a remote daemon
 - Yes
 - No

- Docker can run natively on the following OSes
 - GNU/Linux
 - Microsoft Windows
 - MacOS X
 - FreeBSD
 - AIX

RUNNING OCI IMAGES IN DOCKER

Lesson 3: Running Containers

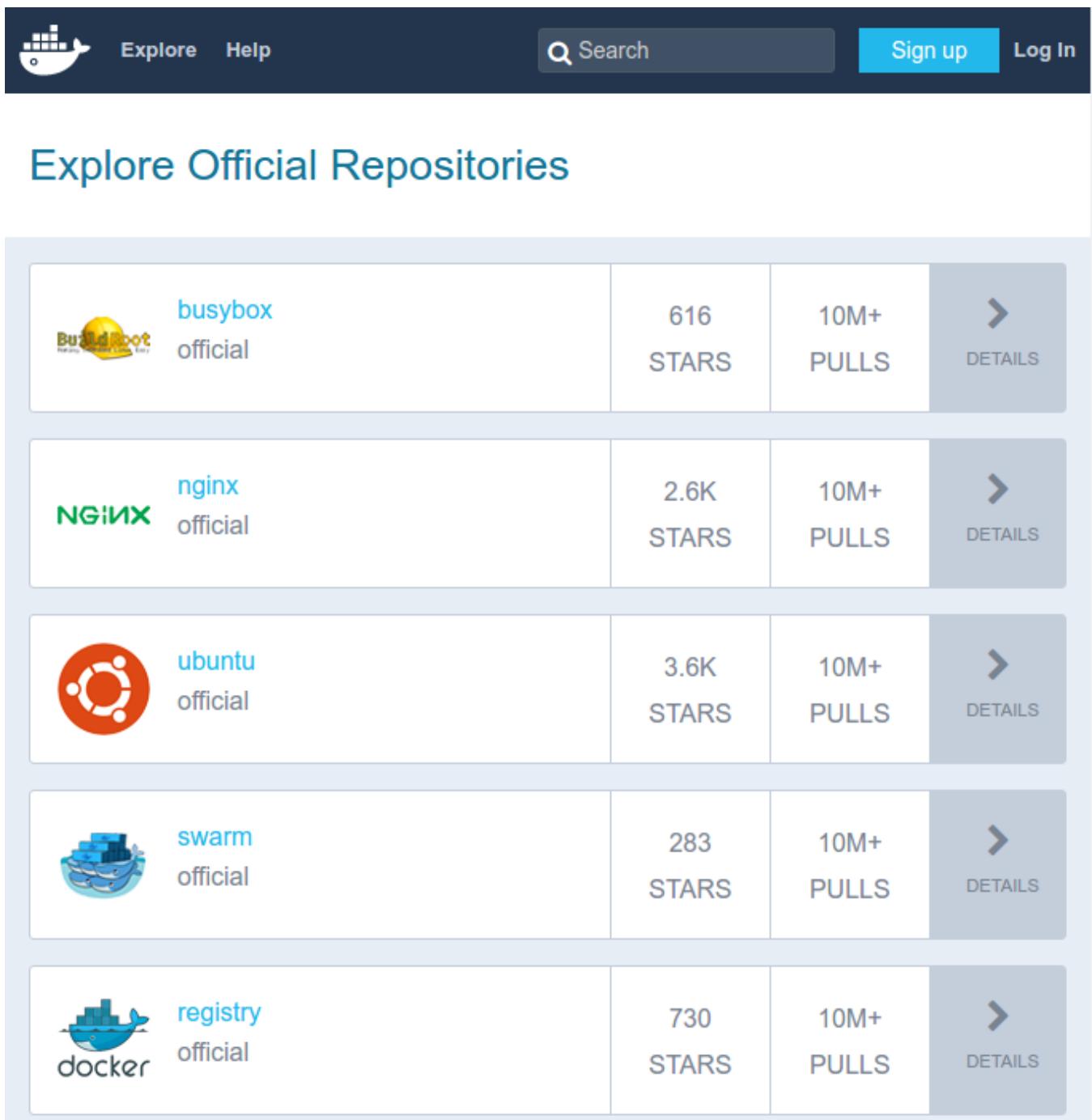
Objectives

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

- find images on DockerHub
- run images as containers
- update a local image from DockerHub
- know the common options for running containers
- stop, start and remove containers
- tune the command executed in a container

The Registry

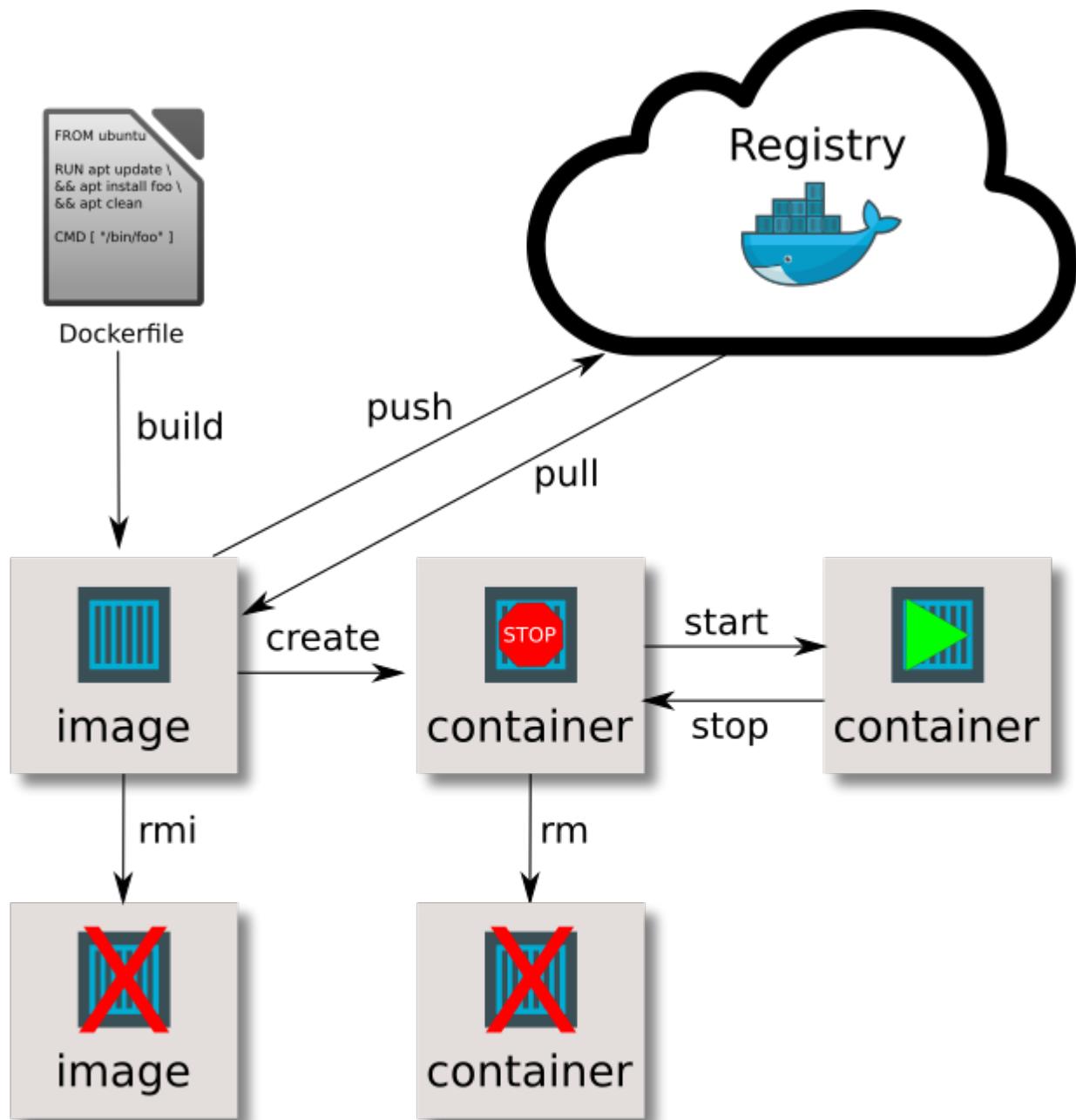
- hub.docker.com by default
- Provides repositories of Docker images



The screenshot shows the Docker Hub interface for exploring official repositories. At the top, there's a dark header bar with a ship icon, 'Explore' and 'Help' links, a search bar containing 'Search', and buttons for 'Sign up' and 'Log In'. Below this, a large blue header says 'Explore Official Repositories'. The main content area displays five repository cards:

Image	Name	Owner	Stars	Pulls	Actions
	busybox	official	616 STARS	10M+ PULLS	DETAILS
	nginx	official	2.6K STARS	10M+ PULLS	DETAILS
	ubuntu	official	3.6K STARS	10M+ PULLS	DETAILS
	swarm	official	283 STARS	10M+ PULLS	DETAILS
	registry	official	730 STARS	10M+ PULLS	DETAILS

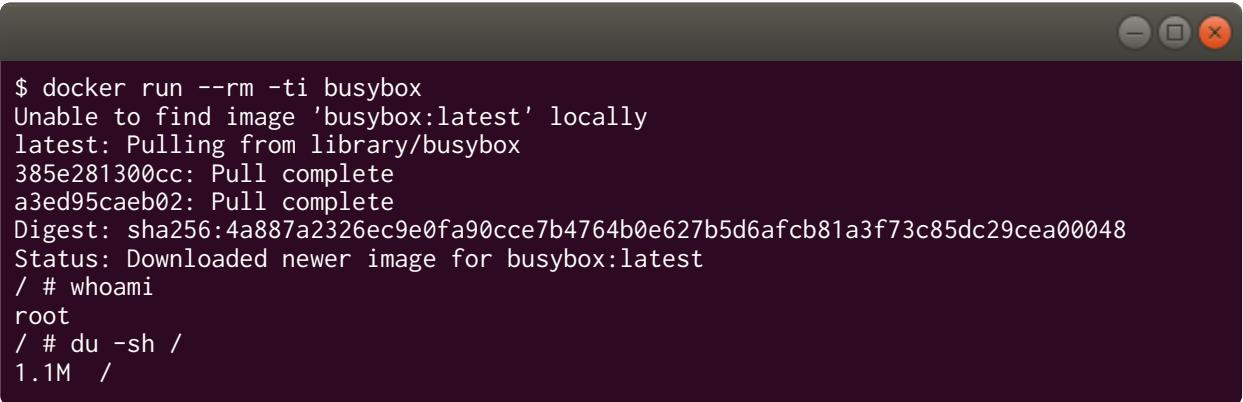
Container lifecycle



run = (pull) + create + start
rm -f = stop + rm

Running images

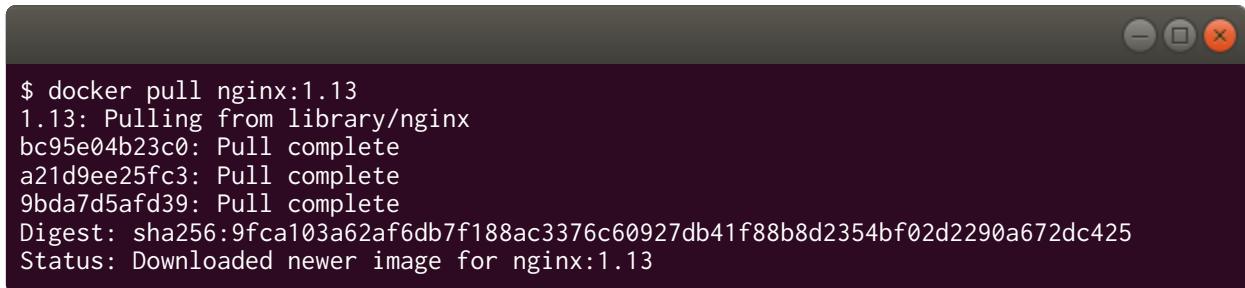
- Automatic pull
- Runs a container from the image



```
$ docker run --rm -ti busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
385e281300cc: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:4a887a2326ec9e0fa90cce7b4764b0e627b5d6afcb81a3f73c85dc29cea00048
Status: Downloaded newer image for busybox:latest
/ # whoami
root
/ # du -sh /
1.1M /
```

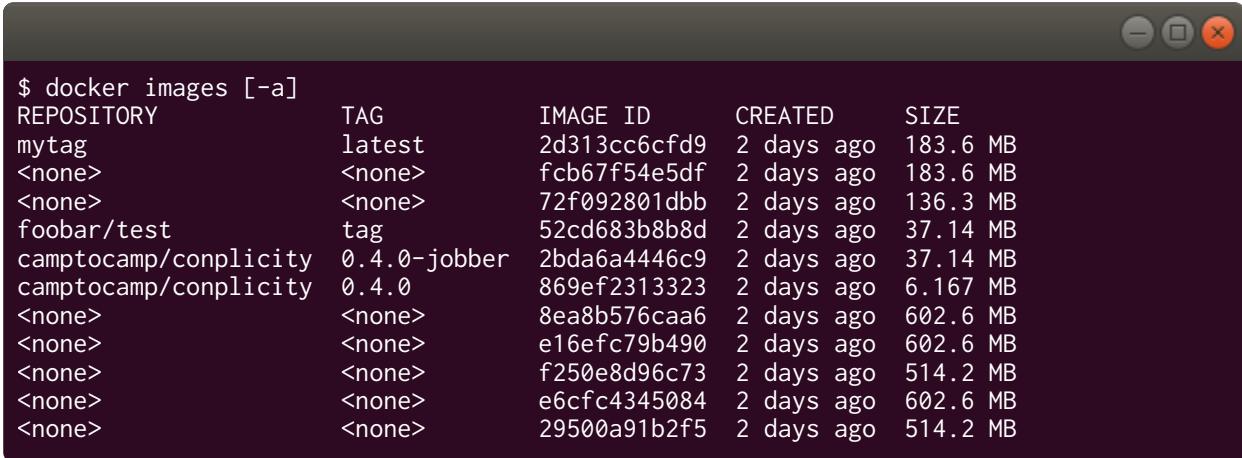
Pulling image

- Force update of local copy



```
$ docker pull nginx:1.13
1.13: Pulling from library/nginx
bc95e04b23c0: Pull complete
a21d9ee25fc3: Pull complete
9bda7d5af39: Pull complete
Digest: sha256:9fca103a62af6db7f188ac3376c60927db41f88b8d2354bf02d2290a672dc425
Status: Downloaded newer image for nginx:1.13
```

Listing images



```
$ docker images [-a]
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
mytag          latest    2d313cc6cf9  2 days ago   183.6 MB
<none>        <none>    fcb67f54e5df  2 days ago   183.6 MB
<none>        <none>    72f092801dbb  2 days ago   136.3 MB
foobar/test    tag      52cd683b8b8d  2 days ago   37.14 MB
camptocamp/conplicity 0.4.0-jobber 2bda6a4446c9  2 days ago   37.14 MB
camptocamp/conplicity 0.4.0       869ef2313323  2 days ago   6.167 MB
<none>        <none>    8ea8b576caa6  2 days ago   602.6 MB
<none>        <none>    e16efc79b490  2 days ago   602.6 MB
<none>        <none>    f250e8d96c73  2 days ago   514.2 MB
<none>        <none>    e6cfcc4345084 2 days ago   602.6 MB
<none>        <none>    29500a91b2f5  2 days ago   514.2 MB
```

Common run options

- `-t` : allocate a pseudo-tty (useful for interactive sessions with `-i`)
- `--rm` : remove container after it is stopped
- `-d` : run in the background (detach)
- `-e <VAR>=<val>` : pass environment variable to container
- `-p <port>[:<port>]` : publish a port on the host
- `-v <volume>[:<volume>]` : mount a volume in the container

Exercise 3.1: explore DockerHub



Objective

- Find an image and run it

Steps

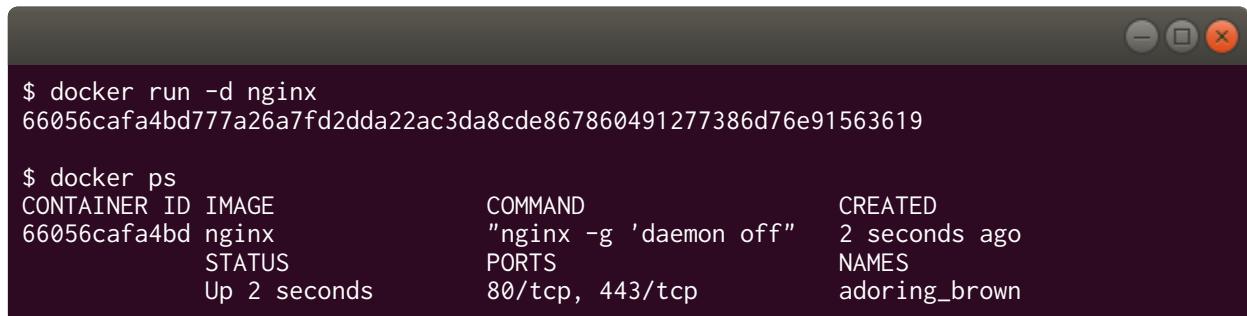
- Using DockerHub or `docker search`, find an image you want to run
- Run the image with `docker run`

Questions

- All images from DockerHub are safe to use
 - Yes
 - No
 - YOLO

ps: Check running containers

- each container has an ID and a name
- the name is automatically generated if not specified

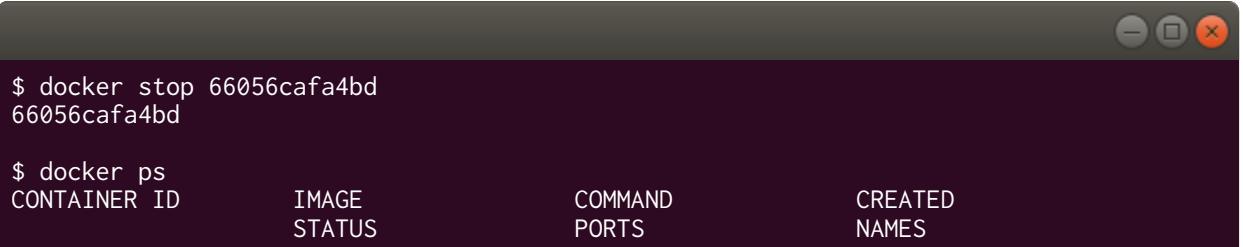


```
$ docker run -d nginx
66056cafa4bd777a26a7fd2dda22ac3da8cde867860491277386d76e91563619

$ docker ps
CONTAINER ID IMAGE COMMAND CREATED
66056cafa4bd nginx "nginx -g 'daemon off'" 2 seconds ago
                  STATUS NAMES
Up 2 seconds      80/tcp, 443/tcp      adoring_brown
```

stop/kill: Stop a container

- `stop` stops a container
- `kill` forces a container to exit



The screenshot shows a terminal window with the following content:

```
$ docker stop 66056cafa4bd
66056cafa4bd

$ docker ps
CONTAINER ID        IMAGE               STATUS            COMMAND           PORTS             CREATED           NAMES

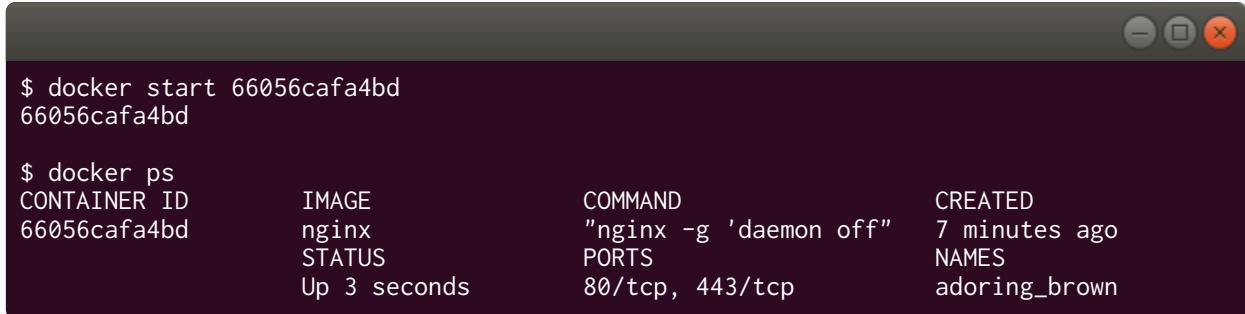
```

Notes:

- the generated name (or set with `--name`) can be used instead of the ID

start: Start a container by ID

- starts an existing container



```
$ docker start 66056cafa4bd
66056cafa4bd

$ docker ps
CONTAINER ID        IMAGE       COMMAND
66056cafa4bd      nginx      "nginx -g 'daemon off"
                      STATUS      PORTS
                          Up 3 seconds           80/tcp,  443/tcp
                                              NAMES
                                              adoring_brown
```

rm: Remove a container

- █ keeps the **image** the container is based on
- █ containers are based on images

```
$ docker rm 66056cafa4bd
Failed to remove container (66056cafa4bd): Error response from daemon:
Conflict, You cannot remove a running container.
Stop the container before attempting removal or use -f

$ docker stop 66056cafa4bd
66056cafa4bd

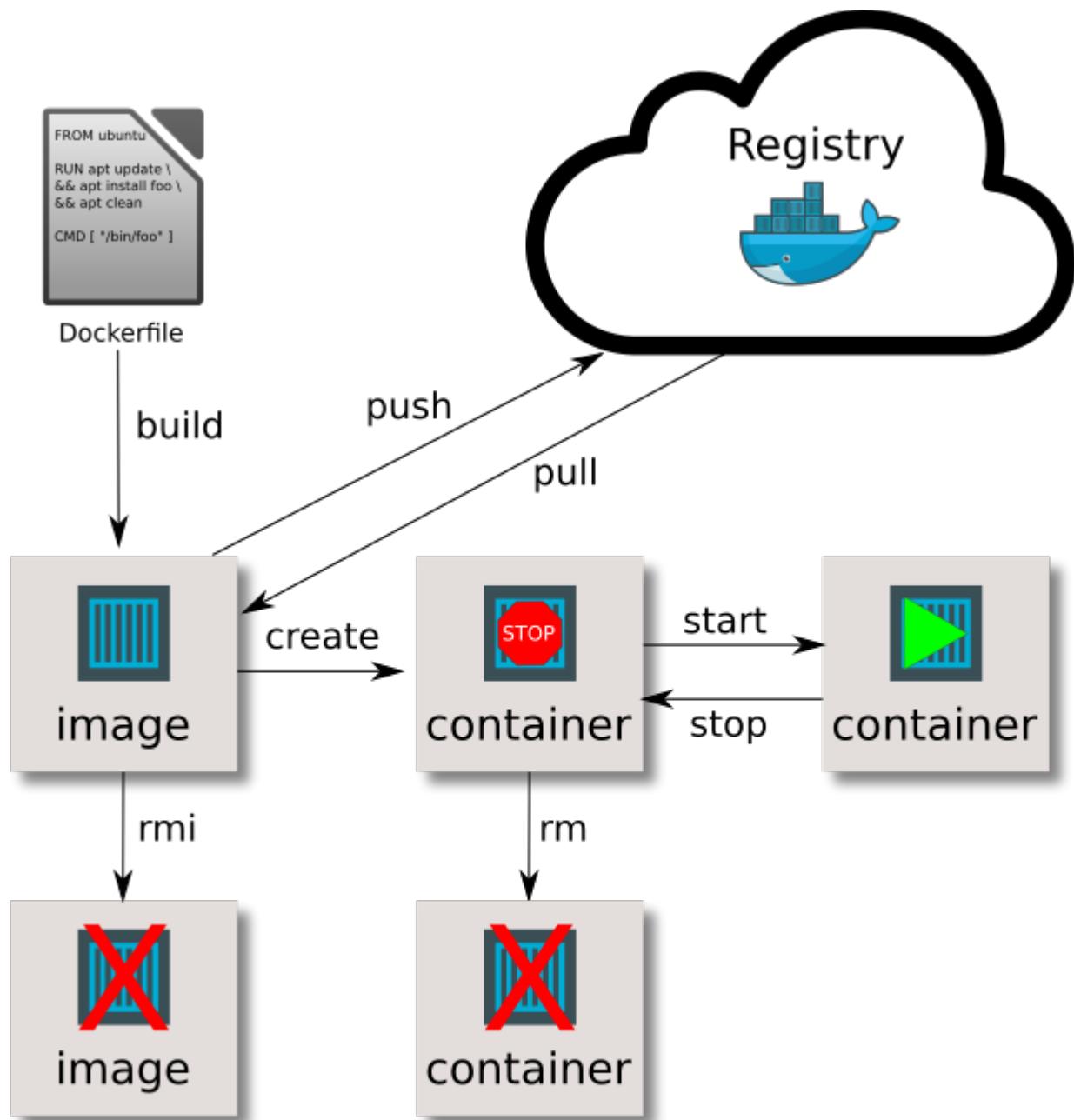
$ docker ps
CONTAINER ID        IMAGE          COMMAND
STATUS             PORTS
CREATED NAMES

$ docker ps -a
CONTAINER ID        IMAGE          COMMAND
66056cafa4bd      nginx          "nginx -g 'daemon off'"
                     STATUS          PORTS
                           Exited (0) 5 seconds ago
                                         CREATED
                                         NAMES
                                         small_northcutt

$ docker rm 66056cafa4bd
66056cafa4bd

$ docker ps -a
CONTAINER ID        IMAGE          COMMAND
STATUS             PORTS
CREATED NAMES
```

Container lifecycle



run = (pull) + create + start
rm -f = stop + rm

Configuration management

Managing software + middleware configuration

Various ways:

- manual (error prone)
- scripts (generated, hard to maintain)

Idempotence



The property of certain operations in mathematics and computer science, that can be applied multiple times without changing the result beyond the initial application. ([Wikipedia](#))

Configuration Management Tools

Managing nodes and services:

- idempotent
- with a declarative language/DSL

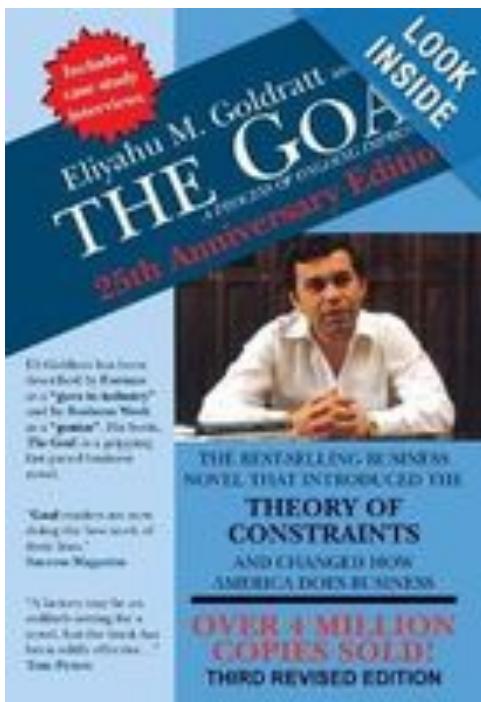
Examples:

- Cfengine
- Puppet
- Chef
- Ansible
- Salt

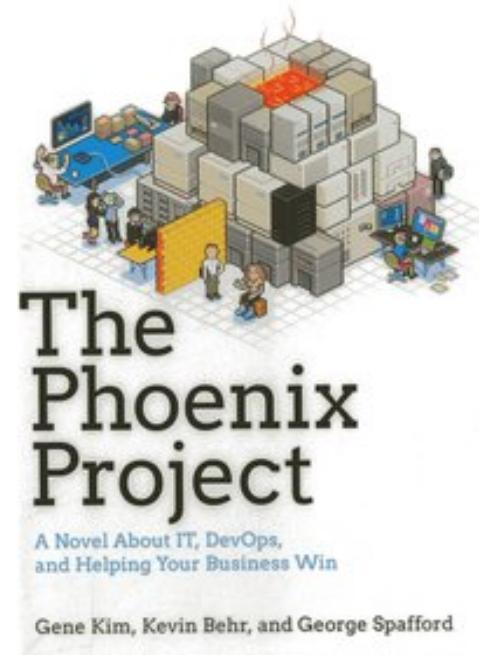
DevOps

- Bridging Operations and Developers
- Three Ways of DevOps:
 - Systems Thinking
 - Amplify Feedback Loops
 - Culture of Continual Experimentation And Learning

More on [IT Revolution](#). Recommended reading:



From the authors of *The Visible Ops Handbook*



Immutable Infrastructure



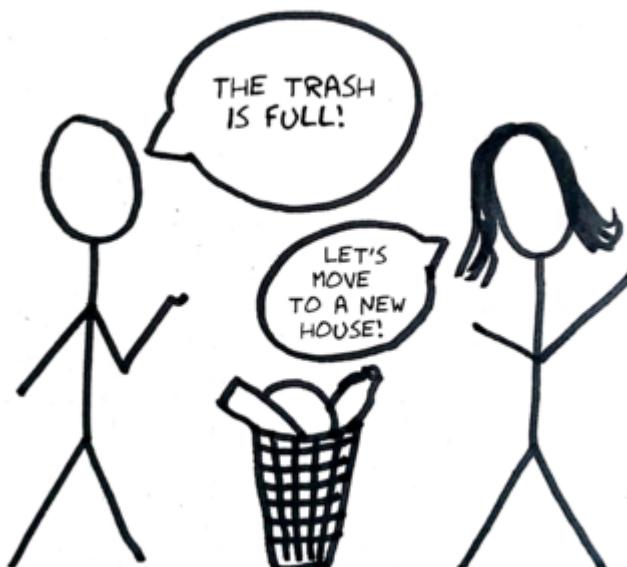
Immutability A pattern or strategy for managing services in which infrastructure is divided into “data” and “everything else”. “Everything else” components are replaced at every deployment, with changes made only by modifying a versioned definition, rather than being updated in-place. ([HighOps](#))

Pros:

- Easier to setup

Cons:

- Requires good processes
- Requires clear code/data separation



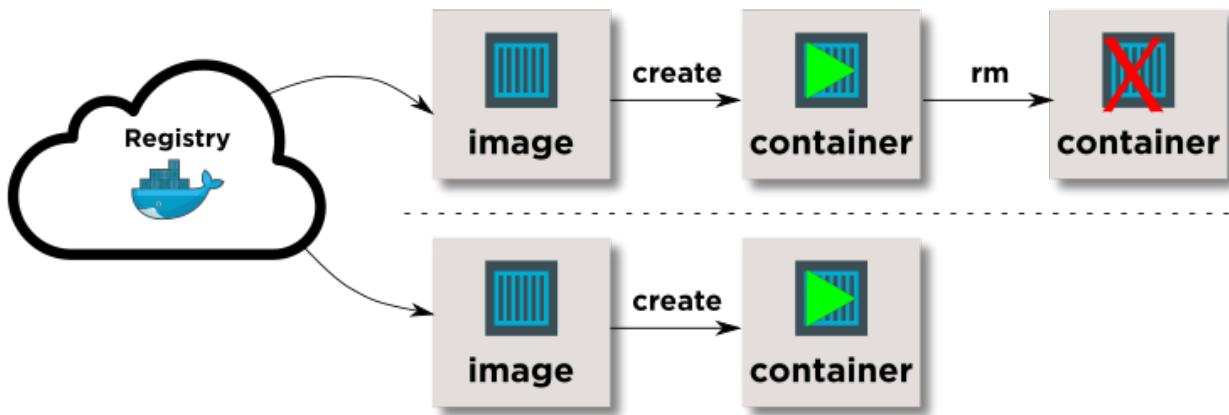
Notes:

- Devops explained: <https://www.niceideas.ch/roller2/badtrash/entry/devops-explained>

Docker and immutability

Docker is made for immutability:

- Manage code in an immutable way. New version ⇒ new image ⇒ new container
- Pass configuration to images (command, environment variables)
- Manage data apart from code (volumes, external services)



Immutability conditions

Immutability requires to separate 3 things:

■ **① code:**

- generic
- versioned
- could be published

■ **② configuration:**

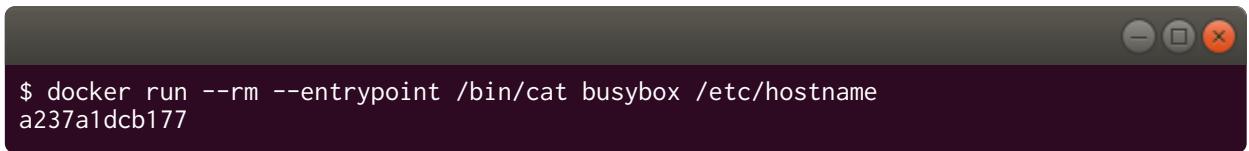
- linked to deployment
- injected into code
- usually private (secrets)

■ **③ data:**

- dynamic/live (cache, journal, session, temp files, etc.)

Entrypoint

- command launched when a container is started
- hardcoded in images
- can be overridden with `--entrypoint`
- default is `sh -c`



```
$ docker run --rm --entrypoint /bin/cat busybox /etc/hostname
a237a1dcbb77
```

A screenshot of a terminal window with a dark background and light text. The window has standard Linux-style window controls at the top right. The terminal output shows a single line of text starting with a dollar sign, followed by a Docker command to run a busybox container with a specific entrypoint and volume mapping.

Command

- the command passed to the entrypoint
- can be specified after the image name

```
$ docker run --rm busybox cat /etc/hostname  
09ea561a8f8e
```

Notes:

- Setting the entrypoint overrides the command

Exercise 3.2: override entrypoint and command



Objective

- Run a container and override its entrypoint/command

Steps

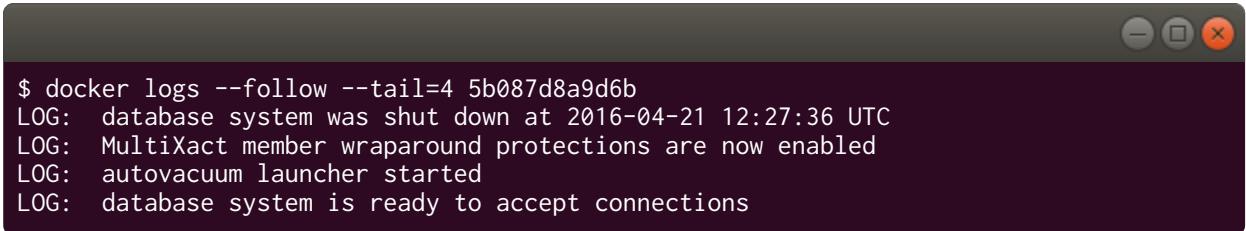
- Choose an image to run
- Run it, overriding either their entrypoint or the command

Questions

- What should the entrypoint map to?
 - the path to the executed process
 - the arguments passed to the executed process
- What should the command map to?
 - the path to the executed process
 - the arguments passed to the executed process

Container logs

- `docker logs` displays container logs since creation (*not* since started)

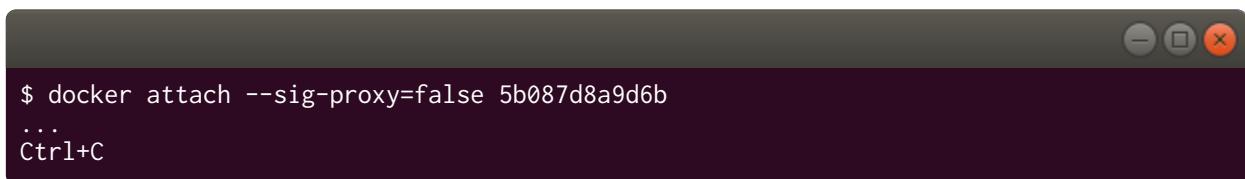


```
$ docker logs --follow --tail=4 5b087d8a9d6b
LOG:  database system was shut down at 2016-04-21 12:27:36 UTC
LOG:  MultiXact member wraparound protections are now enabled
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
```

Attaching to a container

Attach to a detached container:

- use `Ctrl+P Ctrl+Q` to detach again
- use `--sig-proxy=false` so that `Ctrl+C` detaches instead of stopping the container

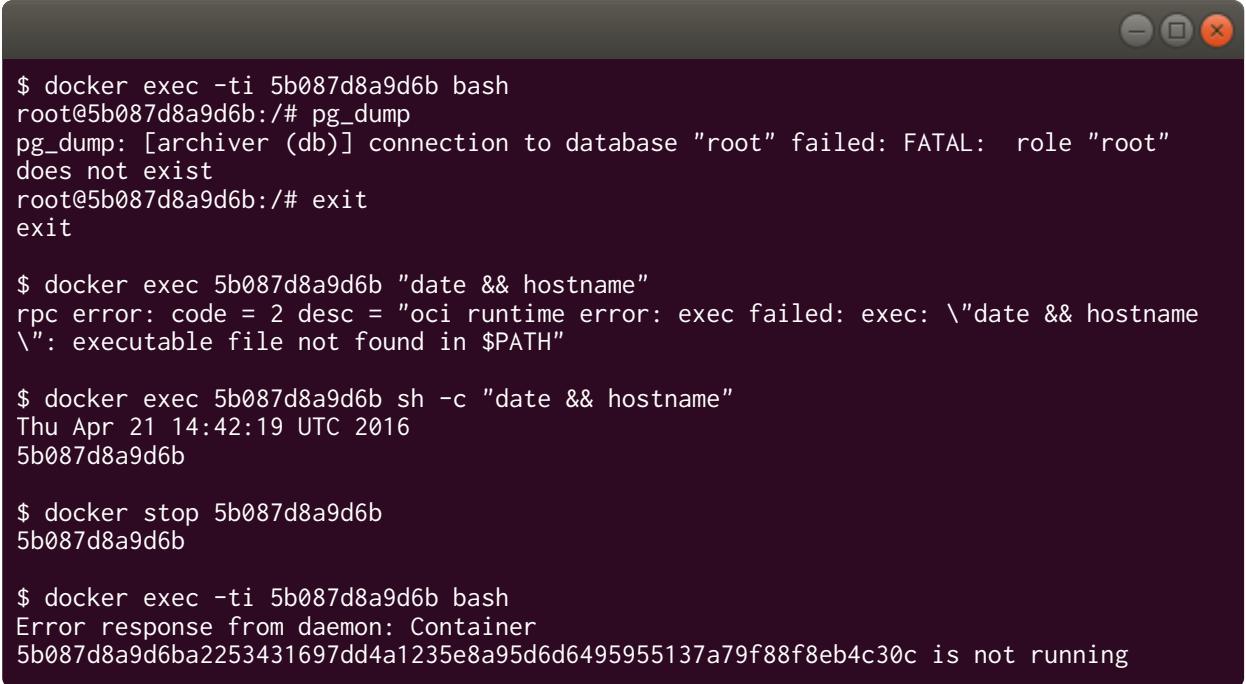


A screenshot of a terminal window with a dark background. The window title bar is dark grey with three small circular icons on the right. The terminal itself has a dark grey header bar with three small circular icons on the right. The main area of the terminal shows the command \$ docker attach --sig-proxy=false 5b087d8a9d6b followed by three dots (...), and then the text Ctrl+C.

```
$ docker attach --sig-proxy=false 5b087d8a9d6b
...
Ctrl+C
```

Execute commands in containers

- `docker exec` executes a command in a *running* container
- commands must not be quoted



```
$ docker exec -ti 5b087d8a9d6b bash
root@5b087d8a9d6b:/# pg_dump
pg_dump: [archiver (db)] connection to database "root" failed: FATAL:  role "root"
does not exist
root@5b087d8a9d6b:/# exit
exit

$ docker exec 5b087d8a9d6b "date && hostname"
rpc error: code = 2 desc = "oci runtime error: exec failed: exec: \\"date && hostname\\": executable file not found in $PATH"

$ docker exec 5b087d8a9d6b sh -c "date && hostname"
Thu Apr 21 14:42:19 UTC 2016
5b087d8a9d6b

$ docker stop 5b087d8a9d6b
5b087d8a9d6b

$ docker exec -ti 5b087d8a9d6b bash
Error response from daemon: Container
5b087d8a9d6ba2253431697dd4a1235e8a95d6d6495955137a79f88f8eb4c30c is not running
```

Transfer files

- `docker cp` allow file transfer between host and container
- Copy file from or to container
- Syntax like `scp`:
 - `docker cp <container>:<SRC> <DST>`
 - `docker cp <SRC> <container>:<DST>`

```
$ docker cp data.json 5b087d8a9d6b:/etc/backend/data.json
$ docker cp 5b087d8a9d6b:/var/log/backend.log /tmp/
```

Special `-` can be used as source or destination to use `STDIN` or `STDOUT`

```
$ ./generate_data.sh | docker cp - 5b087d8a9d6b:/etc/backend/data.json
$ docker cp 5b087d8a9d6b:/var/log/backend.log - | grep ERROR | wc -l
```



only for dev purpose!

Exercise 3.3: Logging in to a running container



Objective

- Log in to a running container

Steps

- Use `docker exec` to start `bash` in running container
- Explore the container in the interactive terminal

Questions

- How is exec different from attach?
 - attach can give you a shell, exec can't
 - exec can give you a shell, attach can't
 - attach lets you see the logs, exec doesn't
 - attach doesn't require the container to run

Environment variables

- Standard for passing parameters to containers (see [12factor.net](#))
- Passed with `-e <KEY>=<VALUE>`

```
$ docker run -e HELLO=world busybox env | grep HELLO  
HELLO=world
```

- Not all applications support environment variables!

Exercise 3.4: Pass environment variables



Objective

- Pass environment variables to a container

Steps

- Run a container and pass environment variables to it
- Check that the variables are properly set

Questions

- What could you use environment variables for?
 - Configuration parameters
 - Secrets
 - Exposing a port
 - Mounting a volume

Checkpoint: Running Containers

Pulling images and running them:

- The default namespace for images points to DockerHub
 - Yes
 - No
- docker run updates images before running them
 - Yes
 - No
- It is safe to clean up images called <none>
 - Yes
 - No
- A destroyed container can be restarted
 - Yes
 - No
- Running containers should be kept up-to-date with a cfgmgmt tool
 - Yes
 - No

BUILDING OCI IMAGES

Lesson 4: Building Images

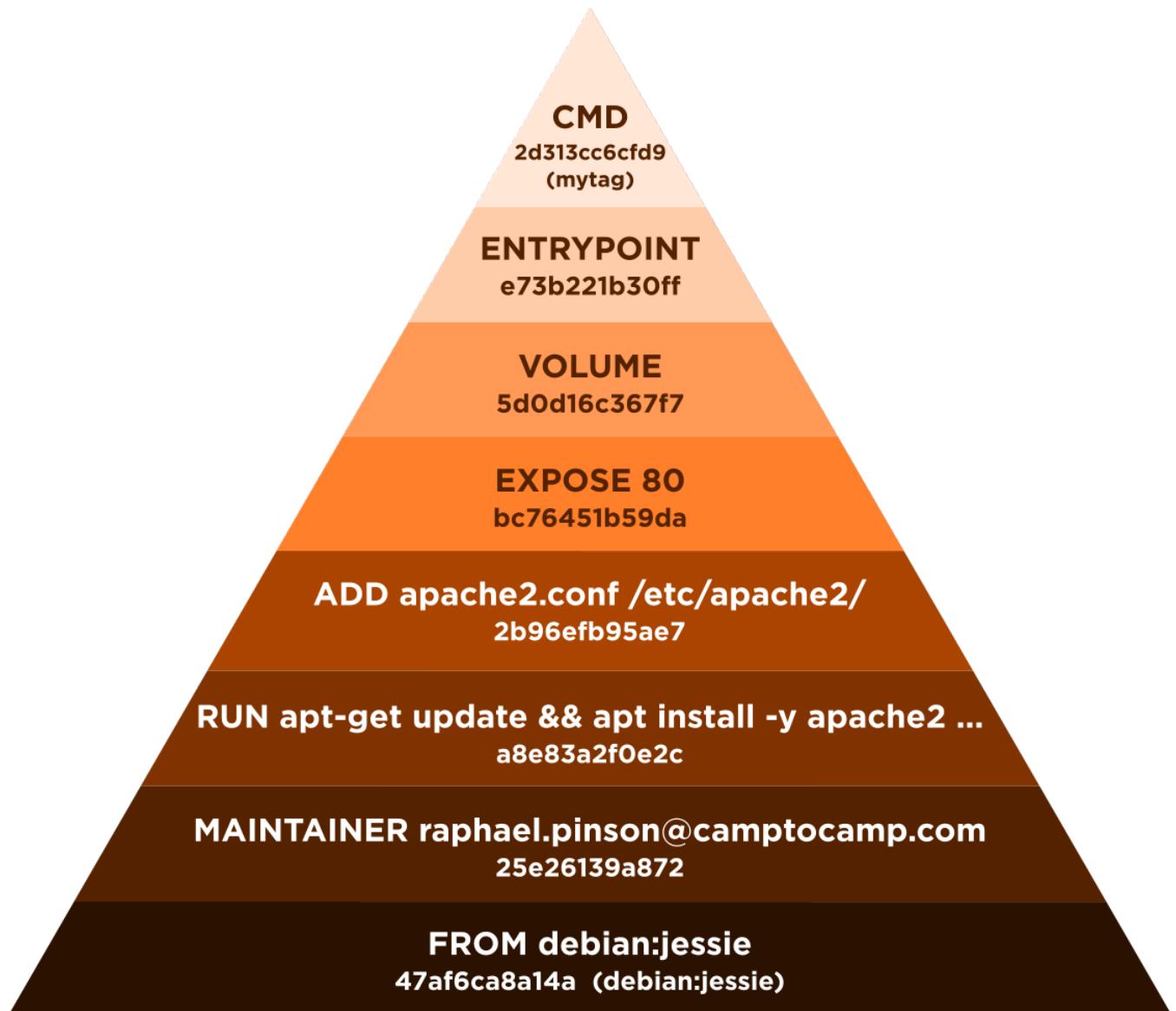
Objectives

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

- understand image layers
- list image layers history
- write a Dockerfile
- build an image from a Dockerfile
- tag an image
- list and remove images

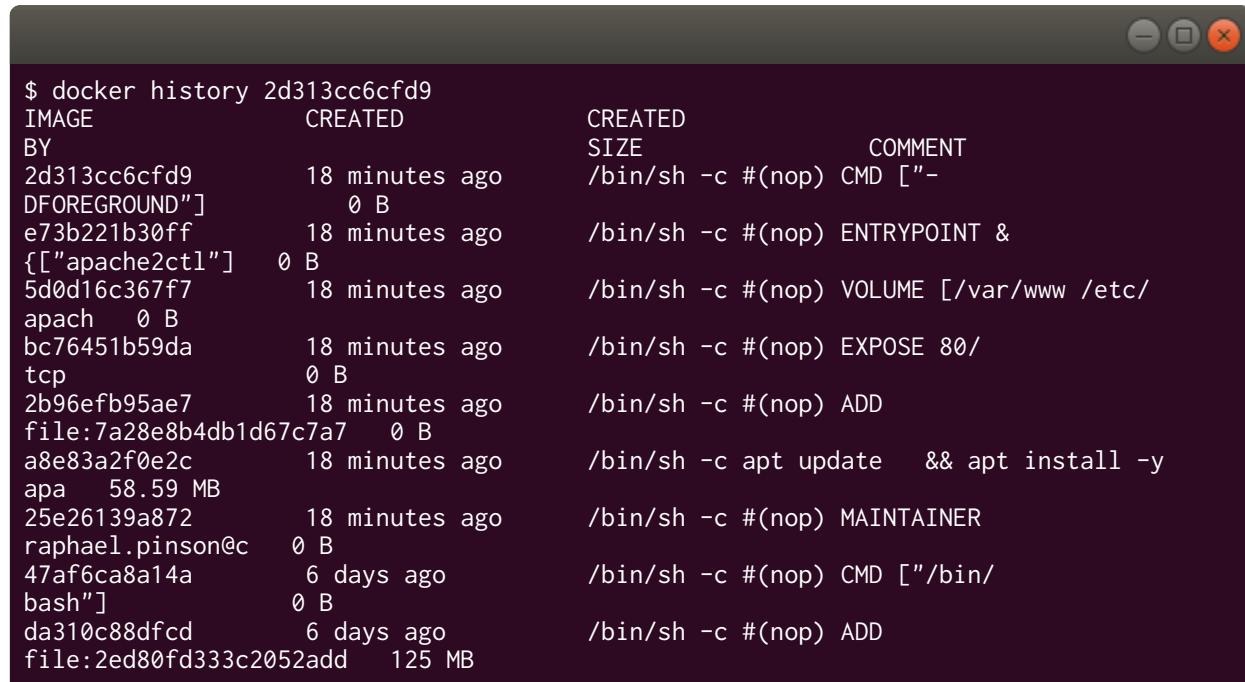
Docker images

Images are made of stacked filesystem layers (using AUFS by default):



Listing image layers

- the `docker history` command lists image layers
- max is 128 layers per image



```
$ docker history 2d313cc6cf9
IMAGE          CREATED      CREATED
BY             18 minutes ago
2d313cc6cf9    18 minutes ago
DFOREGROUND"] 0 B
e73b221b30ff  18 minutes ago
{"["apache2ctl"]} 0 B
5d0d16c367f7  18 minutes ago
apach 0 B
bc76451b59da  18 minutes ago
tcp 0 B
2b96efb95ae7  18 minutes ago
file:7a28e8b4db1d67c7a7 0 B
a8e83a2f0e2c  18 minutes ago
apa 58.59 MB
25e26139a872  18 minutes ago
raphael.pinson@c 0 B
47af6ca8a14a  6 days ago
bash"] 0 B
da310c88dfcd  6 days ago
file:2ed80fd333c2052add 125 MB
```

inspect: Inspecting an image

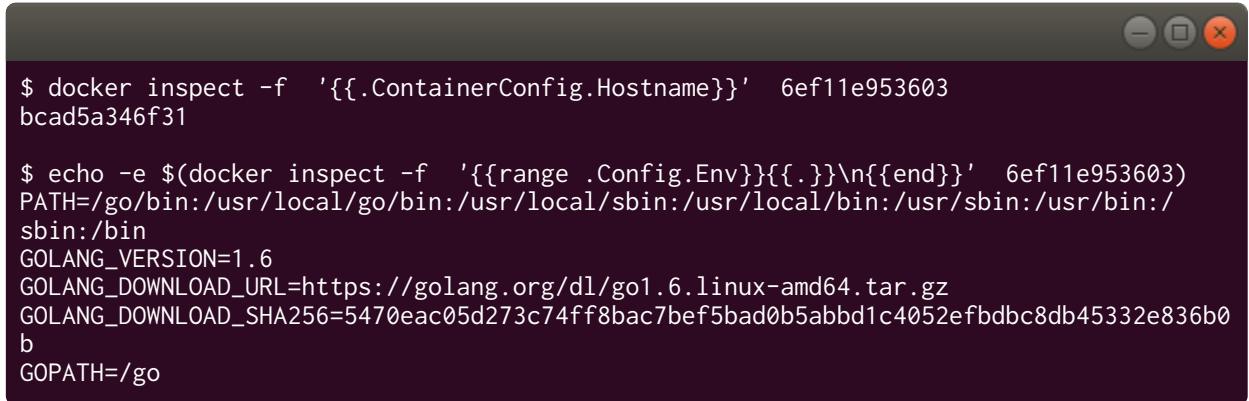
■ `docker inspect <container|image>`



```
$ docker inspect 6ef11e953603
[{"Id": "sha256:6ef11e953603b58d1bdbd4a89b4d6ebb3fe9f349440d61107b1a58f6ffcf50b2", "RepoTags": ["raphink/github_pki:latest"], "RepoDigests": [], "Parent": "sha256:808a0d599148c6e8120d8211db20b9dab908203a7b285f8ee3ef295a12a6fe00", "Comment": "", "Created": "2016-04-14T07:38:19.791832616Z", "Container": "b7883fb19b88b90f6d35064d230ecb7bf5a9eb26677e4c03bd09fc827f29dc4c", "ContainerConfig": {"Hostname": "bcad5a346f31", "Domainname": "", "User": "", "AttachStdin": false, "AttachStdout": false, "AttachStderr": false, "Tty": false, "OpenStdin": false, "StdinOnce": false, "Env": ["PATH=/go/bin:/usr/local/go/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", "GOLANG_VERSION=1.6", "GOLANG_DOWNLOAD_URL=https://golang.org/dl/go1.6.linux-amd64.tar.gz", "GOLANG_DOWNLOAD_SHA256=5470eac05d273c74ff8bac7bef5bad0b5abbd1c4052efbdb8db45332e836b0b", "GOPATH=/go"], "Cmd": ["/bin/sh", "-c", "go-wrapper install"], "ArgsEscaped": true, "Image": "sha256:808a0d599148c6e8120d8211db20b9dab908203a7b285f8ee3ef295a12a6fe00", "Volumes": null, "WorkingDir": "/go/src/app", "Entrypoint": null, "OnBuild": [], "Labels": {}}, "}], }
```

Inspect with templates

- `docker inspect` supports go templates to retrieve specific values



```
$ docker inspect -f '{{.ContainerConfig.Hostname}}' 6ef11e953603
bcad5a346f31

$ echo -e $(docker inspect -f '{{range .Config.Env}}{{.}}\n{{end}}' 6ef11e953603)
PATH=/go/bin:/usr/local/go/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin
GOLANG_VERSION=1.6
GOLANG_DOWNLOAD_URL=https://golang.org/dl/go1.6.linux-amd64.tar.gz
GOLANG_DOWNLOAD_SHA256=5470eac05d273c74ff8bac7bef5bad0b5abbd1c4052efbdb8db45332e836b0
b
GOPATH=/go
```

The Dockerfile format

```
FROM debian:jessie

MAINTAINER raphael.pinson@camptocamp.com

RUN apt update \
    && apt install -y apache2 \
    && rm -rf /var/lib/apt/lists/*

ADD apache2.conf /etc/apache2/

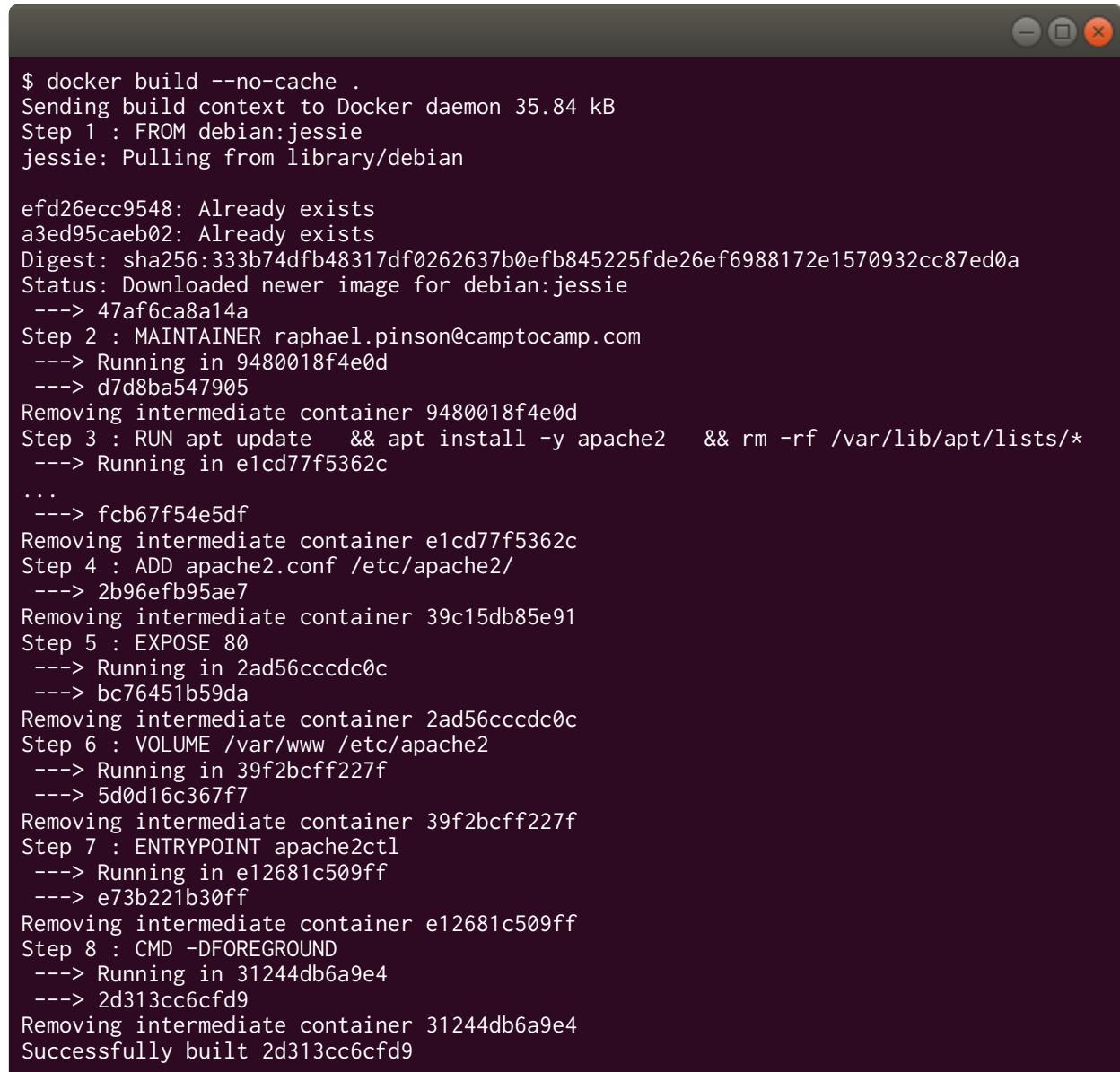
EXPOSE 80

VOLUME ["/var/www", "/etc/apache2"]

ENTRYPOINT ["apache2ctl"]
CMD ["-DFOREGROUND"]
```

The Build command

- Builds the image using the local Dockerfile
- Caches intermediary images by default
- Use `--pull` to ensure the base image is up-to-date



```
$ docker build --no-cache .
Sending build context to Docker daemon 35.84 kB
Step 1 : FROM debian:jessie
jessie: Pulling from library/debian

efd26ecc9548: Already exists
a3ed95caeb02: Already exists
Digest: sha256:33b74dfb48317df0262637b0efb845225fde26ef6988172e1570932cc87ed0a
Status: Downloaded newer image for debian:jessie
    --> 47af6ca8a14a
Step 2 : MAINTAINER raphael.pinson@camptocamp.com
    --> Running in 9480018f4e0d
    --> d7d8ba547905
Removing intermediate container 9480018f4e0d
Step 3 : RUN apt update && apt install -y apache2 && rm -rf /var/lib/apt/lists/*
    --> Running in e1cd77f5362c
...
    --> fcb67f54e5df
Removing intermediate container e1cd77f5362c
Step 4 : ADD apache2.conf /etc/apache2/
    --> 2b96efb95ae7
Removing intermediate container 39c15db85e91
Step 5 : EXPOSE 80
    --> Running in 2ad56cccdc0c
    --> bc76451b59da
Removing intermediate container 2ad56cccdc0c
Step 6 : VOLUME /var/www /etc/apache2
    --> Running in 39f2bcff227f
    --> 5d0d16c367f7
Removing intermediate container 39f2bcff227f
Step 7 : ENTRYPOINT apache2ctl
    --> Running in e12681c509ff
    --> e73b221b30ff
Removing intermediate container e12681c509ff
Step 8 : CMD -DFOREGROUND
    --> Running in 31244db6a9e4
    --> 2d313cc6cf9
Removing intermediate container 31244db6a9e4
Successfully built 2d313cc6cf9
```

Dockerfile: FROM

Sets the base image to use.

```
FROM debian:jessie
```

- `FROM scratch` (0-size virtual image)
- `FROM nginx` (official image)
- `FROM debian:jessie` (official image with tag)
- `FROM camptocamp/git` (personal repository)
- `FROM camptocamp/github_pki:latest` (personal repository with tag)

Building the Dockerfile: FROM

**FROM debian:jessie
47af6ca8a14a (debian:jessie)**

Notes :

```
$ docker build --no-cache .
Sending build context to Docker daemon 35.84 kB
Step 1 : FROM debian:jessie
jessie: Pulling from library/debian
efd26ecc9548: Already exists
a3ed95caeb02: Already exists
Digest: sha256:333b74dfb48317df0262637b0efb845225fde26ef6988172e1570932cc87ed0a
Status: Downloaded newer image for debian:jessie
--> 47af6ca8a14a
```

Lab 4.1: Create a Dockerfile



Objective

- Create a Dockerfile with a FROM directive and build it

Steps

- Choose a base image from DockerHub
- Create a directory and a Dockerfile
- Add a FROM directive to the Dockerfile
- Build the image using docker build
- Run the image with docker run

Questions

- Did this exercise generate a new image hash?
 - Yes
 - No
- Why or why not?
- Can the same Dockerfile generate different images?
 - Yes
 - No

Dockerfile: MAINTAINER

Sets the maintainer of the image

```
MAINTAINER raphael.pinson@camptocamp.com
```

MAINTAINER is deprecated, use LABEL maintainer instead:

```
LABEL Maintainer="raphael.pinson@camptocamp.com" Vendor="Camptocamp"
```

Building the Dockerfile: MAINTAINER

MAINTAINER raphael.pinson@camptocamp.com

25e26139a872

FROM debian:jessie
47af6ca8a14a (debian:jessie)

Notes:

```
Step 2 : MAINTAINER raphael.pinson@camptocamp.com
--> Running in 9480018f4e0d
--> d7d8ba547905
Removing intermediate container 9480018f4e0d
```

Lab 4.2: Add labels



Objective

- Add a maintainer to the Dockerfile and rebuild

Steps

- Edit your Dockerfile to add a `LABEL` directive
- Rebuild the image
- Use `docker history` to check the layers

Questions

- What could image labels be used for?
 - Adding metadata to the image
 - Downloading image updates
 - Autoconfiguring other services

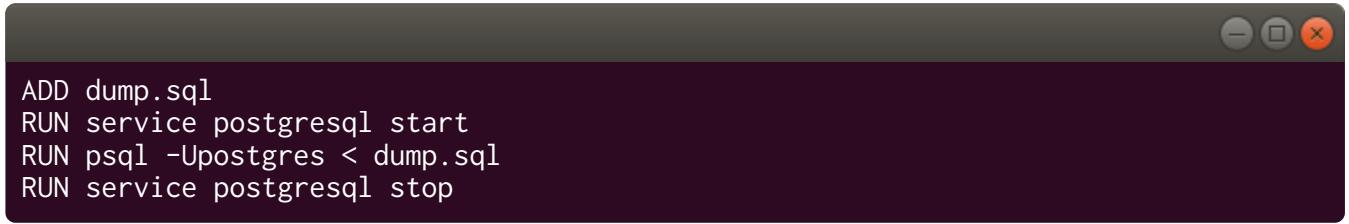
Dockerfile: RUN

- Runs a command in the container
- Generates a new layer image

```
RUN apt update \  
    && apt install -y apache2 \  
    && rm -rf /var/lib/apt/lists/*
```

Notes:

- it is recommended to purge cache systems (e.g. apt) after installing to reduce the image size
- each `RUN` directive generates a layer, commands return and do not keep running. The following does NOT work:



```
ADD dump.sql  
RUN service postgresql start  
RUN psql -Upostgres < dump.sql  
RUN service postgresql stop
```

Building the Dockerfile: RUN

```
RUN apt-get update && apt install -y apache2 ...
a8e83a2f0e2c
```

```
MAINTAINER raphael.pinson@camptocamp.com
25e26139a872
```

```
FROM debian:jessie
47af6ca8a14a (debian:jessie)
```

Notes :

```
Step 3 : RUN apt update    && apt install -y apache2    && rm -rf /var/lib/apt/
lists/*
---> Running in e1cd77f5362c
...
---> fcb67f54e5df
Removing intermediate container e1cd77f5362c
```

Minimizing Image Size

- Each layer takes space
- `RUN` steps must seek to minimize delta

E.g.:

```
RUN apt-get update          # Adds a cache layer  
RUN apt-get -y install apache2 # Adds a software install layer  
RUN apt-get clean            # Adds 0-size layer "hiding" cache files
```

VS

```
RUN apt-get update \  
    && apt-get -y install apache2 \  
    && apt-get clean           # Adds only a software install layer
```

Lab 4.3: Add a RUN statement



Objective

- Add a RUN statement to your Dockerfile and rebuild it

Steps

- Edit your Dockerfile and add a `RUN` command (e.g. install a package)
- Rebuild your image
- Run your image and check the effect of the `RUN` command (e.g. software is installed)

Questions

- Can you use RUN when using an image FROM scratch?
 - Yes
 - No

Dockerfile: ADD

- Add a local file to the container
- Generates a new layer image
- Both `COPY` and `ADD` add files to layers
- `ADD` performs "magic" actions (download, untar, etc.)

```
ADD apache2.conf /etc/apache2/
```

Notes:

- `COPY` is a more simple version of `ADD` which doesn't automatically download/extract files

Building the Dockerfile: ADD



Notes :

```
Step 4 : ADD apache2.conf /etc/apache2/  
---> 2b96efb95ae7  
Removing intermediate container 39c15db85e91
```

Build Context

The Docker service, not the client, builds the image:

- Docker sends its context (all files in local directory) to the service before build
- Heavy files will be sent for every build
- A `.dockerignore` file can be added to exclude files from being sent



You cannot access files outside of the current directory

Lab 4.4: Add an ADD statement



Objective

- Add an ADD statement to your Dockerfile and rebuild it

Steps

- Edit your Dockerfile and add an `ADD` directive (e.g. a configuration file or a script)
- Rebuild your image
- Run your image and check the effect of the `ADD` directive

Questions

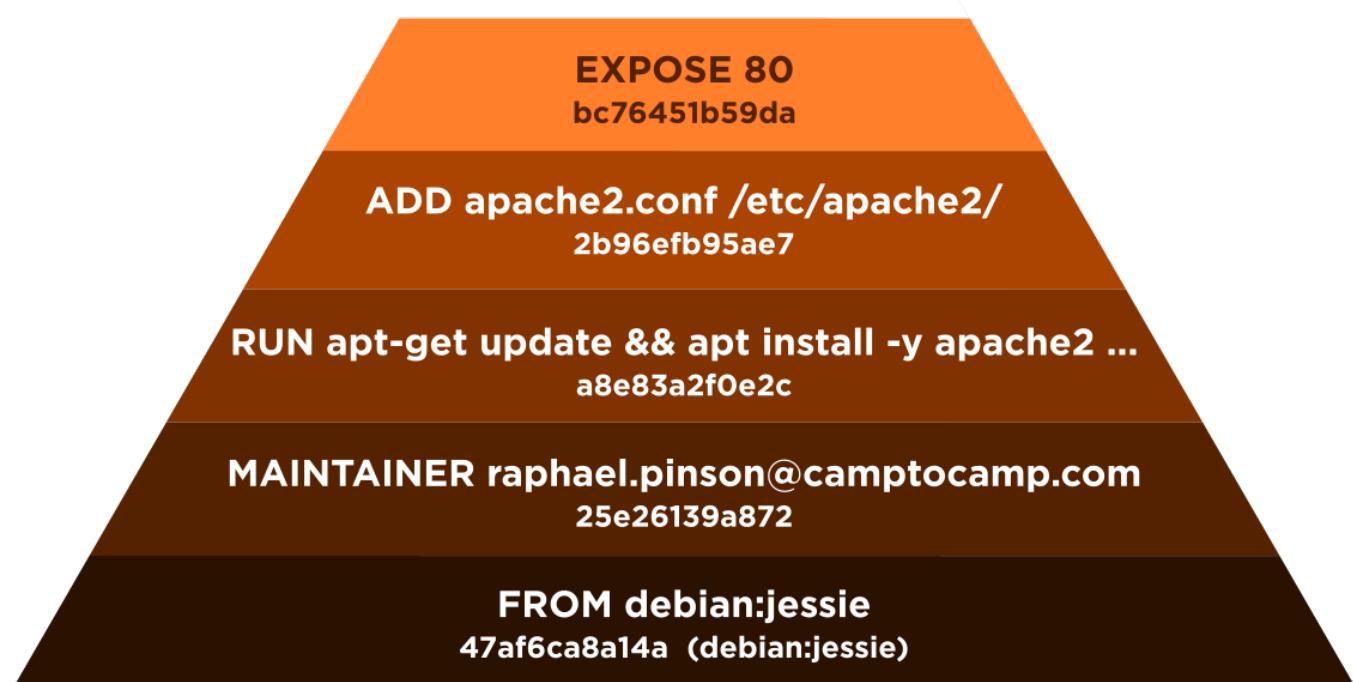
- What could you do to limit the amount of layers?
 - Use wildcards with ADD
 - Pass several files to ADD
 - Use a tarball with ADD
 - Use COPY instead of ADD

Dockerfile: EXPOSE

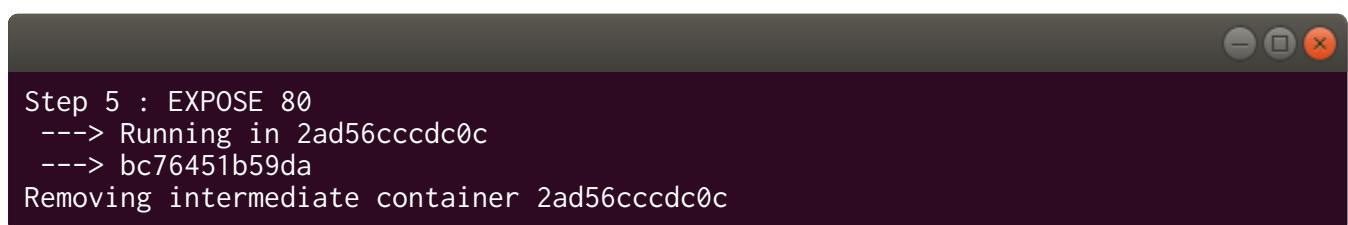
- Declares publishable ports in the container
- Does not imply that ports will be published

```
EXPOSE 80
```

Building the Dockerfile: EXPOSE



Notes :



```
Step 5 : EXPOSE 80
--> Running in 2ad56cccdc0c
--> bc76451b59da
Removing intermediate container 2ad56cccdc0c
```

Lab 4.5: Expose ports



Objective

- Expose ports in your Dockerfile and rebuild

Steps

- Edit your Dockerfile and add an `EXPOSE` directive
- Rebuild your image
- Run your image with `--publish-all` and check that the ports are published (using `netstat` for example)

Questions

- Can an image expose more than one port?
 - Yes
 - No
- Is it safe to expose insecure ports?
 - Yes
 - No

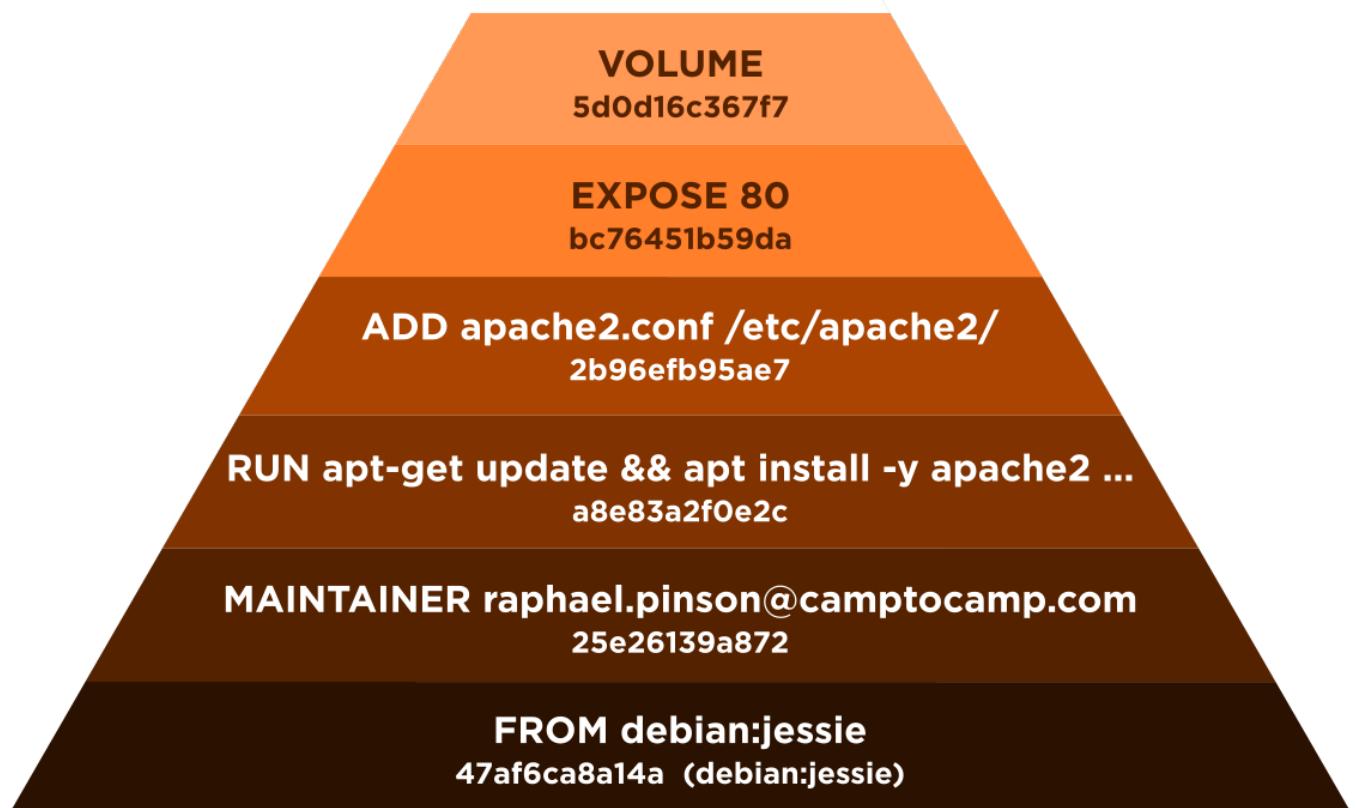
Dockerfile: VOLUME

- Sets volumes to be automatically exported for binding
- Creates separated volumes (similar to partitions) on the host system

```
VOLUME ["/var/www", "/etc/apache2"]
```

- These volumes will be automatically mounted in containers at runtime

Building the Dockerfile: VOLUME



Notes:

```
Step 6 : VOLUME /var/www /etc/apache2
--> Running in 39f2bcff227f
--> 5d0d16c367f7
Removing intermediate container 39f2bcff227f
```

VOLUME: Order Matters

- Docker mounts declared volumes immediately during build
- Files placed in a declared volume during build:
 - are available during build time (unless another volume is declared on top)
 - will be **lost** in the final image

```
FROM debian

VOLUME /usr/share/apache2

RUN apt-get update \
    && apt-get -y install apache2 \
    && apt-get clean

# All files in /usr/share/apache2 are lost in the final image!
```

Lab 4.6: Declare volumes



Objective

- Declare volumes in your Dockerfile and rebuild

Steps

- Edit your Dockerfile to export volumes
- Rebuild your image
- Run a container from the image
- Inspect the container to see which volumes were created

Questions

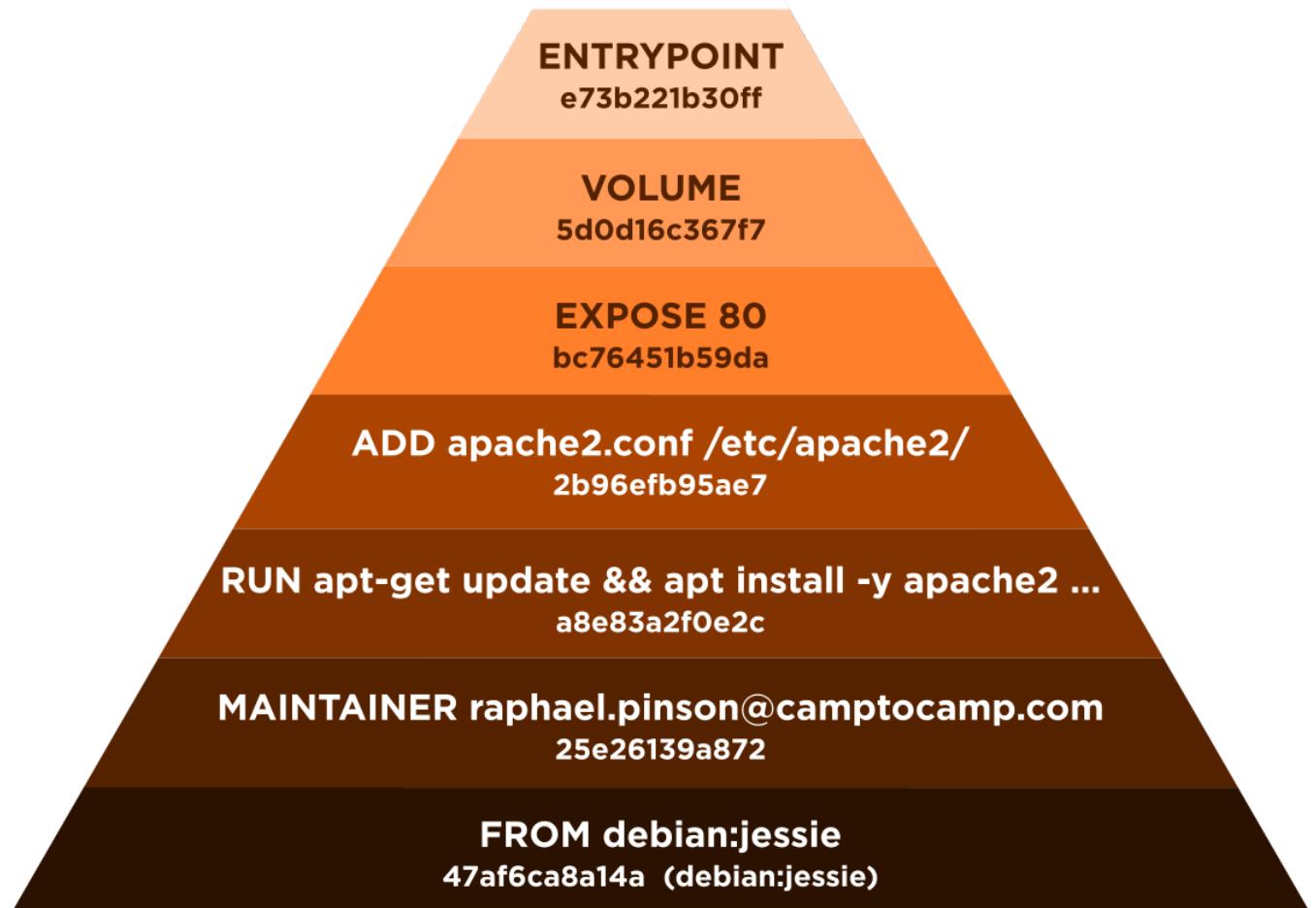
- How can you check if the volume was created?
 - Use mount on the host
 - Use mount in the container
 - Use inspect on the image
 - Use inspect on the container

Dockerfile: ENTRYPOINT

Sets the entrypoint of the container

```
ENTRYPOINT ["apache2ctl"]
```

Building the Dockerfile: ENTRYPOINT



Notes:

```
Step 7 : ENTRYPOINT apache2ctl
--> Running in e12681c509ff
--> e73b221b30ff
Removing intermediate container e12681c509ff
```

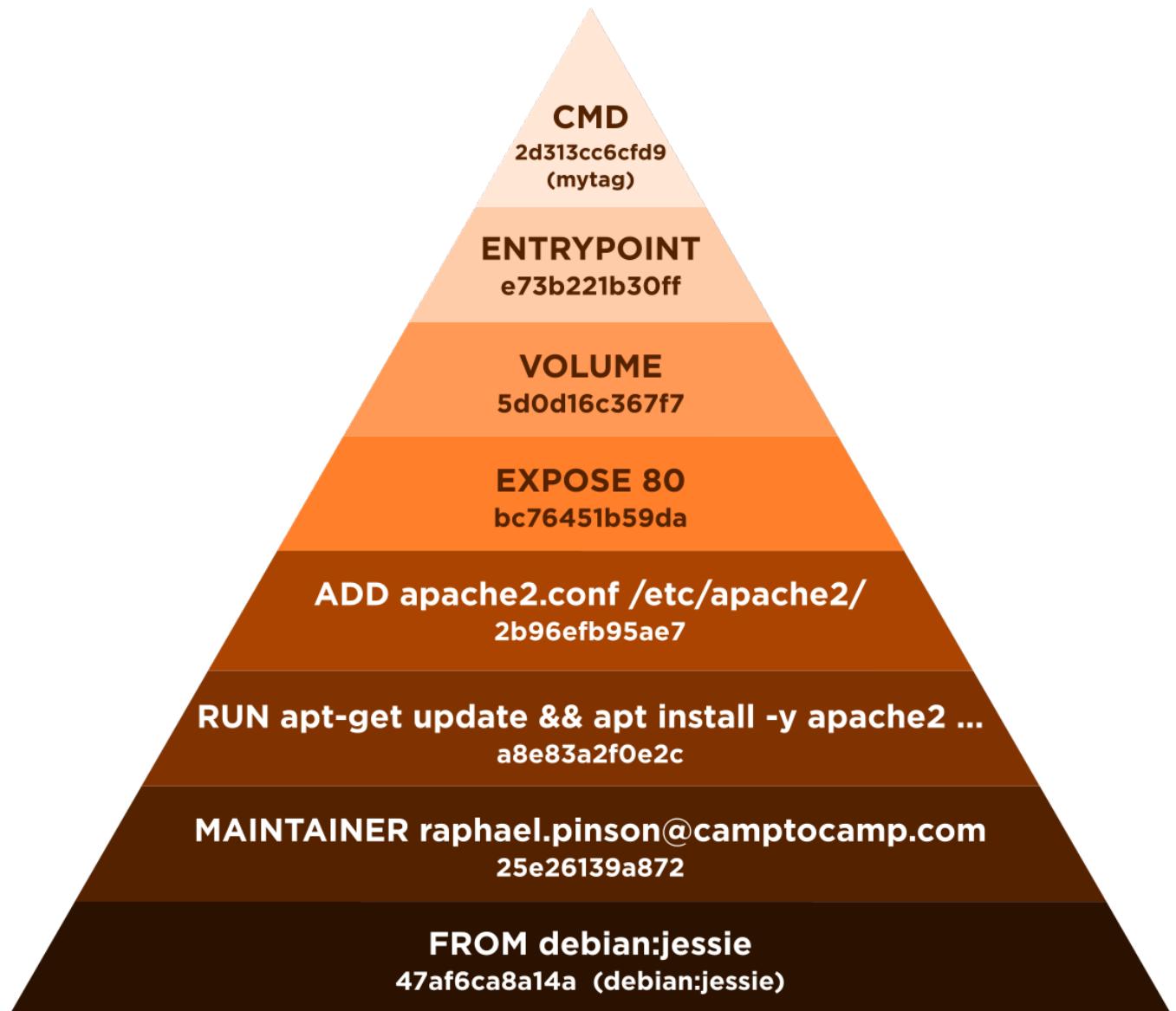
Dockerfile: CMD

Set the default command of the container (arguments to the entrypoint)

```
CMD ["-DFOREGROUND"]
```

ENTRYPOINT vs CMD formats and interaction: <https://docs.docker.com/engine/reference/builder/#understand-how-cmd-and-entrypoint-interact>

Building the Dockerfile: CMD



Notes:

```
Step 8 : CMD -DFOREGROUND
--> Running in 31244db6a9e4
--> 2d313cc6cf9
Removing intermediate container 31244db6a9e4
Successfully built 2d313cc6cf9
```

Lab 4.7: Set the entrypoint and default command



Objective

- Set the entrypoint and default command in your Dockerfile and rebuild

Steps

- Edit your Dockerfile and add `ENTRYPOINT` and `CMD` directives
- Rebuild your image
- Run the image and check that the command is properly run

Questions

- Where would you place a flag that should not be overridden?
 - In the ENTRYPOINT
 - In the CMD

Creating an Entrypoint

If you need to customize the service initialization, create a shell script that ends with an `exec`:

```
#!/bin/sh

# Configure program to log to stdout, not a log file!
echo "Log = stdout" >> /etc/prg.conf

# Make the entrypoint behave like the program itself
exec /usr/bin/prg "$@"
```

`exec` ensures the service:

- keeps pid 1
- receives all signals (including KILL), so it shuts down correctly.

More Actions

Other possible actions before launching the program:

- Wait for another service
- Build html static files / seed cache
- Service discovery / dynamic configuration

Use a **flexible docker entrypoint**:

```
#!/bin/bash
DIR=/docker-entrypoint.d
if [[ -d "$DIR" ]]
then
    /bin/run-parts --verbose "$DIR"
fi
exec "$@"
```

```
COPY docker-entrypoint.sh /
COPY docker-entrypoint.d/* /docker-entrypoint.d/
ONBUILD COPY docker-entrypoint.d/* /docker-entrypoint.d/
ENTRYPOINT ["/docker-entrypoint.sh", "/opt/puppetlabs/puppet/bin/mcollectived"]
CMD ["--no-daemonize"]
```

Wait for Another Service

See <https://github.com/vishnubob/wait-for-it>:

- Minimal dependencies
- Waits for services to be ready (database, etc.)

```
while :  
do  
    echo > /dev/tcp/$HOST/$PORT  
    if [ $? -eq 0 ]; then  
        break  
    fi  
done
```

Injecting env vars in Config Files

Environment variables are the most flexible way to tune a container's behavior but not all programs support environment variables

■ In Dockerfile:

- `RUN sed -i ...`
- `RUN envsubst`
- `ADD apache2.conf from context`
- Augeas

■ At runtime:

- run sed/envsubst, Augeas in the entrypoint shell script



Environment variables declared in the Dockerfile (with ENV) are available during runtime as well.

Checking container status

Docker can monitor health of application by running a command in the container.

When a `HEALTHCHECK` is specified in `Dockerfile`:

- a new `State.Health` section is added to the container status
- this command is run regularly.

Based on exit code of command define in `HEALTHCHECK` container health status is updating.

Health status:

- `starting`: no healthcheck done
- `healthy`: healthcheck command exit with '0'
- `unhealthy`: healthcheck command exit with '1'

Checking container status

Command must exit with 0 or 1. Behaviour with other exit code is not defined.

```
HEALTHCHECK --interval=1s CMD \
  curl -s -o /dev/null -w "%{http_code}" localhost:8080 | grep 200 || exit 1
```

- `--interval`: time to wait after complete check and at startup
- `--timeout`
- `--start-period`: Ignore failed check during this period
- `--retries`: Consecutive failed check

Health status is just a metadata, container will not be restarted, this is the job of orchestrators.

Checking container status

How to check health of application?

- check current microservice
- don't check external service
- failed check trigger restart and consume resources
- implement a dedicate route: `/health`

Failure of external service

If external service is down, container stay healthy but return errors.

- Container should not be restarted
- But should not be used

Container is alive but not *ready* to use.



Lab 4.8: Define healthcheck



Objective

- Monitor health of application

Steps

- Find a command to check health of app
- Edit your Dockerfile and add a HEALTHCHECK
- Rebuild your image
- Delete `data.json` in container and check health status

Questions

- Is container restarted?
 - yes
 - no
- Can healthcheck impact performances?
 - yes
 - no
- Can I monitor non HTTP services?
 - yes
 - no
- What happen if my healthcheck is based on external services?
 - impact stability of host
 - better error handling
 - useless restart are performed

Multi Stage Build

When tools needed at build time are not required for runtime:

- Compiler
- Libs
- Source code

Multi stage build allows to create several images and copy files as artifacts from one image to another. Multiple named `FROM` stages.

```
FROM golang AS builder
RUN go build ...
...
FROM scratch
COPY --from=builder ...
```

The last image is the result.

External images can also be used as a stage:

```
COPY --from=nginx:latest /etc/nginx/nginx.conf /nginx.conf
```

Lab 4.9: Use multi stage build



Objective

- Use multi stage build to create a smaller image

Steps

- Edit your Dockerfile and add a new stage
- Copy files from one stage to another
- Rebuild your image
- Run the image and check that only files copied from one stage to another are present in the image.

Questions

- How to choose the base image for the last stage?
 - Contains headers/SDK needed for building
 - Contains CA cert bundle
 - Contains an interpreted language (NodeJs, Python, PHP)
 - Contains a compiled language (C, Rust, Go)
 - Contains runtime dependencies

Labs solutions

You can find `Dockerfile` for backend and frontend [here](#).

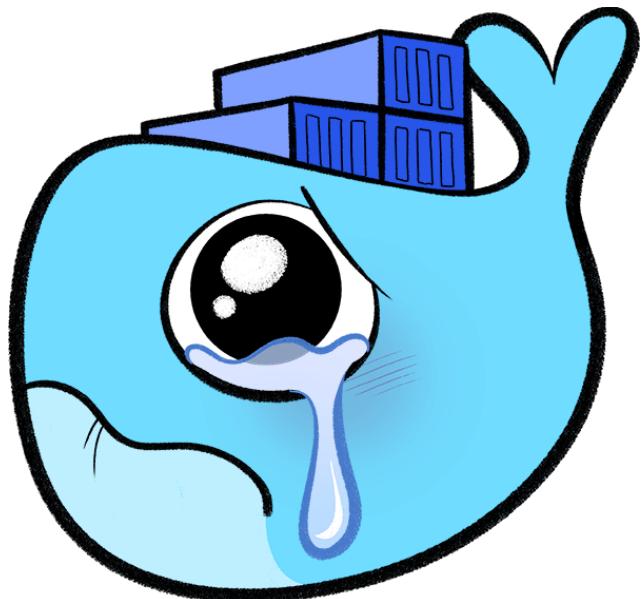
Debugging build

On build errors,

- run a container with the hash of the previous step
- and use a shell as entrypoint and set `stdin_open` and `tty` to `true`

```
Sending build context to Docker daemon 113.7kB
Step 1/10 : FROM python:3-slim
--> 3b8cf061fbb0
Step 2/10 : RUN cat /nonexist
--> Running in 09b6d88969fc
cat: /nonexist: No such file or directory
The command '/bin/sh -c cat /nonexist' returned a non-zero code: 1
```

```
$ docker run -ti --entrypoint="" 3b8cf061fbb0 /bin/bash
```



Debugging at runtime

Start container with shell entrypoint:

```
$ docker run -ti --entrypoint="" myimage:2.0 /bin/bash
```



If no shell is present in image, use a busybox container

```
$ docker run -d -ti -v busybox:/bin busybox
$ docker run --rm -ti --entrypoint="" -v busybox:/usr/local/bin:ro myimage:2.0 /usr/local/bin/busybox
```



Listing and removing images

```
$ docker images -a
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
<none>        <none>    5d0d16c367f7    2 days ago   183.6 MB
<none>        <none>    2b96efb95ae7    2 days ago   183.6 MB
<none>        <none>    e73b221b30ff    2 days ago   183.6 MB
<none>        <none>    bc76451b59da    2 days ago   183.6 MB
mytag          latest    2d313cc6cf9    2 days ago   183.6 MB
<none>        <none>    a8e83a2f0e2c    2 days ago   183.6 MB
<none>        <none>    25e26139a872    2 days ago   125 MB

$ docker rmi 5d0d16c367f7
Failed to remove image (5d0d16c367f7): Error response from daemon:
  conflict: unable to delete 5d0d16c367f7 (cannot be forced)
  - image has dependent child images
```

Image Cleanup

`docker system` provides information and cleaning tools for images and containers.

- `docker system df` displays size stats
- `docker system prune` cleans up Docker data (unused images and containers):
 - `--all` also removes non-dangling images
 - `--force` does not prompt for confirmation

Tagging images

- images are identified by a hash
- a tag is an alias to an image hash
- tags can be added at build time

```
$ docker build --no-cache -t foobar/test:latest .
Sending build context to Docker daemon 18.07 MB
Step 1 : FROM blacklabelops/jobber:latest
--> 09c12e8e220f
Step 2 : ADD complicity /usr/bin/
--> 52cd683b8b8d
Removing intermediate container 854a4bb5738d
Successfully built 52cd683b8b8d
```

- or afterwards

```
$ docker tag 52cd683b8b8d mylocaltest
$ docker tag foobar/test:latest myaliasstag
```

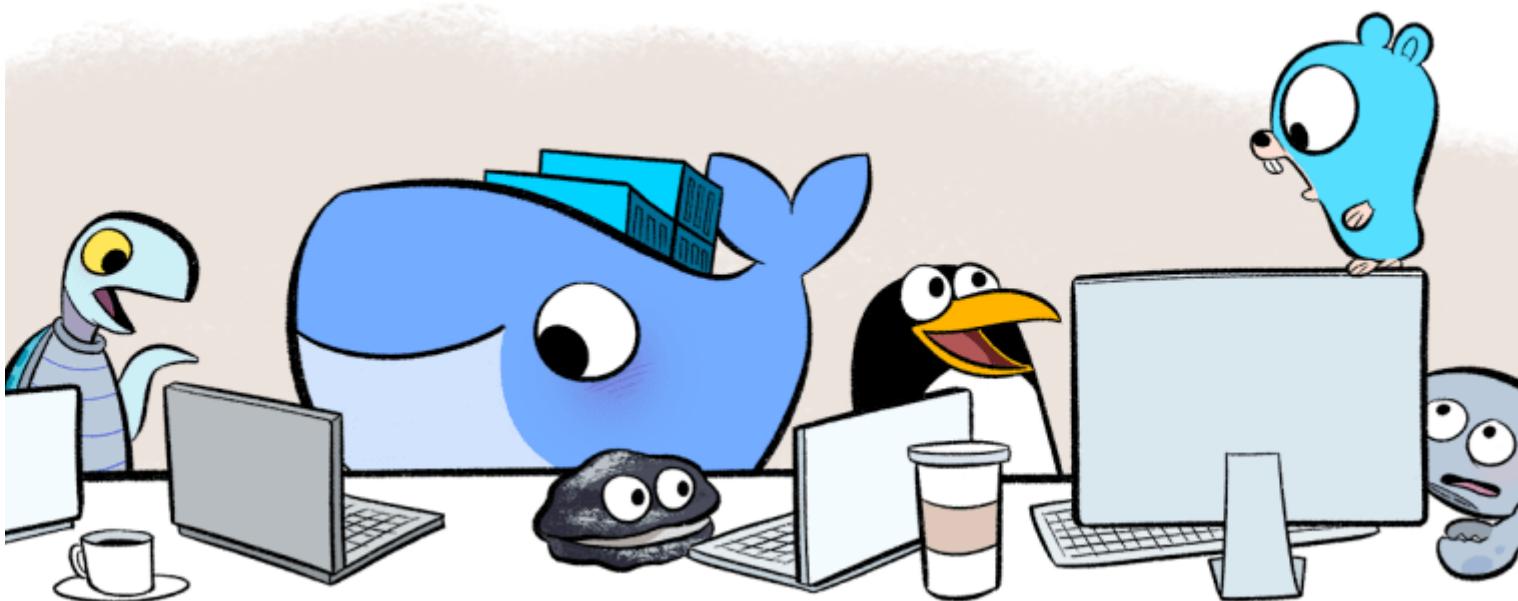
OCI Specification



Docker images follow the OCI (Open Container Initiative) specification. See <https://opencontainers.org/>

Other build/management tools for OCI images:

- **Buildah** builds containers without docker service
- **Img** builds containers without docker service
- **Skopeo** manipulates docker images between registries
- **Kaniko** builds docker images within a kubernetes cluster or container



Checkpoint: Building Images

- Each directive in the Dockerfile produces a new layer
 - Yes
 - No

- Dockerfile supports the following directives
 - RUN
 - DOWNLOAD
 - BUILD
 - ONBUILD
 - FROM
 - EXIT

- How can you filter information returned by docker inspect?
 - Use jq
 - Use jgrep
 - Use the --format flag with Go templating

- Exposing ports automatically publishes them on the node
 - Yes
 - No

- Declaring volumes is useful to
 - Improve IO performance
 - Share volumes between containers
 - Store data in the cloud

APPLICATION ARCHITECTURE

Lesson 5: Application Architecture

Objectives

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

- understand microservice architecture principles
- create stateless apps
- understand security principles: arbitrary user, unprivileged ports, mount rights, capabilities, etc.

Microservices

No strict definition.



Microservice architecture arranges an application as a collection of loosely coupled services ([Wikipedia > Microservices](#))

- Application split into minimal components
- Loosely-coupled

Advantages:

- easier to scale in an orchestrator
- easier to test
- easier to reuse
- possible to scale/update a specific component
- possible to use multiple programming languages
- possible to distribute development

Microservices: Architecture

- one process per microservice ⇒ per container
- granularity, balance between monolith and lambda functions:
 - start overhead
 - code complexity
 - ext lib dependencies vs internal services dependencies
- isolate stateful components



A Stateful application depends on an internally managed state (usually local files) to function.

Avoiding Stateful Components



Stateful components don't scale well unless planned for it (e.g. cluster of stateful components with an ordonancer component)

Use external states:

- Database
- Key/value storage
- Object Storage

Use a distributed file system (slower):

- NFS
- GlusterFS
- CephFS



On Kubernetes, use *StatefulSet* objects.

Security

Containers isolate applications, but there are still security concerns:

- Container Engine User
- Container User
- Container Network
- Container Ports
- Mounts
- Capabilities

Security: Container Engine User

Docker is a daemon:

- running as root
- containers are launched as root (can be demoted later)
- no link between user session and user container
- easy privilege escalation on node (unless the OS prevents it, e.g. CentOS/RedHat)

Security: Rootless Containers

Podman

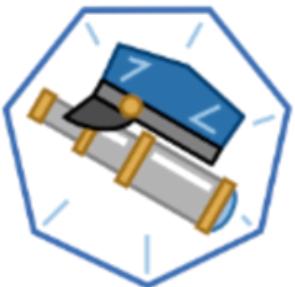
- not a daemon
- runs as user (requires `sudo` to be root)
- associated tools
 - Buildah/Kaniko to build
 - Skopeo



podman



buildah



skopeo

Security: Container User

- By default: `root`
- Demoting container: `docker run --user=$UID`
- Default user can be set in `Dockerfile` with a `USER` instruction (doesn't create user, create it with a `RUN` directive)
- Limits potential Kernel exploits



RedHat OpenShift starts containers with pseudo-random UIDs (>1000000)

Security: Container Network

Options:

- Bridge (default)
- None
- Container
- Host

```
# Use another container's network
$ docker run --net=container:deadbeef ...

# Use Host network
$ docker run --net=host tcpdump ...
```



Using the Host Network in a container is highly discouraged.

Security: Container Ports

When not running as root:

- Cannot bind ports >1024 in container
- Need to configure software to run on unprivileged ports



Since Docker runs as root, ports can still be *published* on privileged ports on host.

Security: Mounts

By default, Docker allows to mount any host path:

- possible to mount `/etc` in RW
- any Docker user can escalate privileges on host



RedHat and CentOS prevent this with default SELinux rules.
Use `:z` and `:Z` options in this case.

When not running as root:

- user permissions matter on mount points
- prefer using named Docker volumes, where Docker manages rights properly on disk

Security: Capabilities

Runtime privileges for Linux containers. Can be set with:

- `--cap-add`: add specific privileges
- `--cap-drop`: remove specific privileges
- `--privileged`: grant many privileges and tune SELinux or AppArmor rules for the container

```
# Run with all capabilities except MKNOD
$ docker run --cap-add=ALL --cap-drop=MKNOD ...
```

See the [list of capabilities in the Docker documentation](#)



Running containers as privileged is highly discouraged.

Checkpoint: Application Architecture

- Application should have one microservice per...
 - method
 - functionality
 - client
 - container
- Which of these are stateful applications?
 - An application storing data in AWS S3
 - An application storing data in PostgreSQL
 - An application storing data in SQLite
 - An application storing data in local XML files
- What is the default network for containers?
 - Host
 - None
 - Bridge
 - Container

APPLICATION DEVELOPMENT

Lesson 6: Application Development

Objectives

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

- Quickly deploy, patch and redeploy applications
- Debug applications running in a container
- Format application logs

Developping with Containers

Pros

- SDK without installing on dev machine
- Standard dev environment (Container Image)
- Local services (database, backend, etc.)
- Acceptance testing (dynamic environment creation)

Challenges

- Code synchronization
- Recompilation
- Ptrace Attachment
- Accessing Application

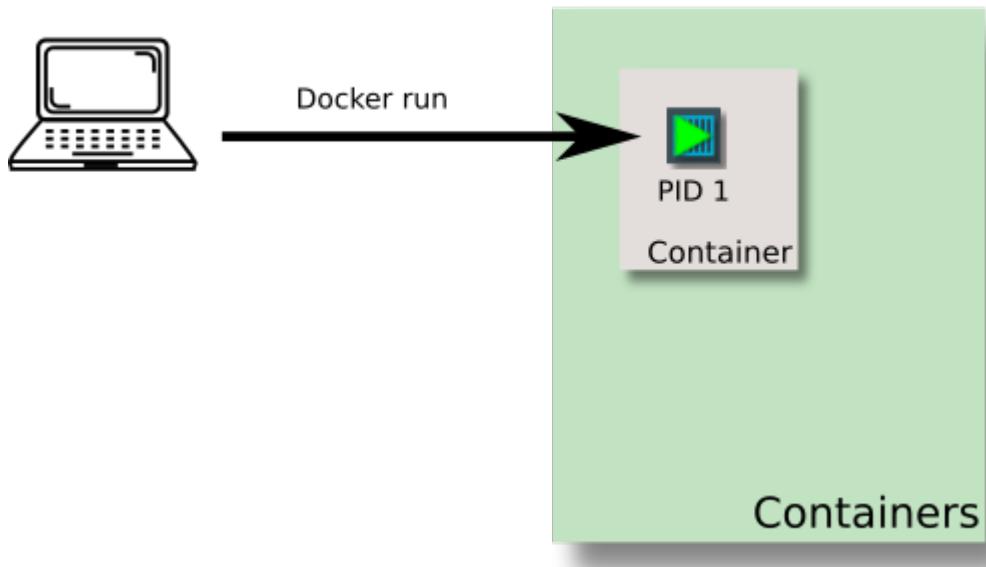
Debug Features

What we want for a full development environment:

- Set breakpoints
- Step by step execution
- Inspect variables
- Evaluate expressions in context
- Inject/replace code in containers

Link Workstation to Container

Challenge: access application (network, files, stack trace, etc.)



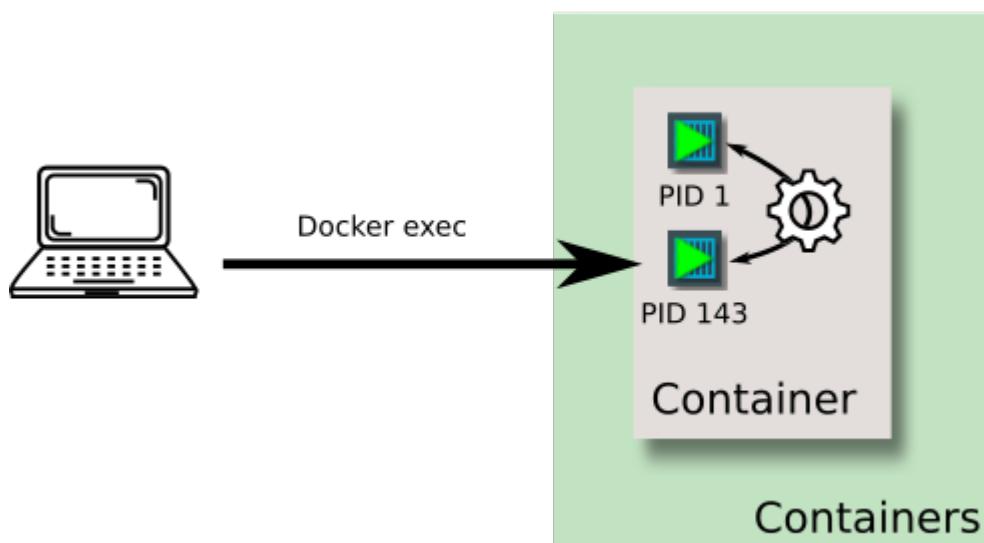
When running containers, isolation prevents communication between the app in the container and the desktop IDE

Debugger need to access source code.

Launch Debugger Inside Container

Debug tools can be run manually within containers:

- Use terminal based interface
- Run in containers as a new process
- Access interface via `docker exec`



Debugger Inside container

- Better accessibility
- Easy to setup

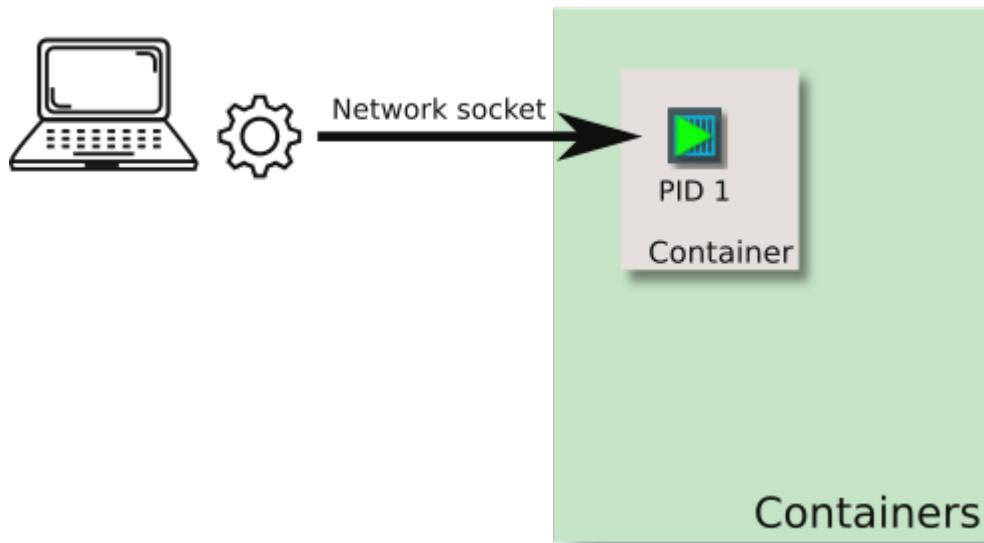
Implementations:

- Python: `pdb`/`pdb++`
- Go: `delve` (requires specific compile flags)

Debugger Outside container

You can also use a graphical Debugger:

- Graphical interface
- Runs on workstation
- Access via network socket
- Container exposes debug port
- IDE initiates connection to container port



Debugger Outside container

- Requires access to container port
- Better ergonomics

Implementations:

- Python: [PyCharm](#), [VSCode](#)
- Go: [VSCode](#)
- Java: [VSCode](#)

Configure App for Debug

Debug is not enabled by default when running or building applications. There are some additional steps to be able to debug an application:

- Build flags/parameters
- Load additional modules/libraries
- Set breakpoints in code

Python

- With pdb/pdb++, you have to load debug lib and add breakpoint in code.
- With VSCode, you need to install the `debugpy` module on system, no app modifications.

Go

- Binaries need to be built with the `-gcflags "all=-N -l"` build flags

Java

- JVM include a debug server. You can enable debug with the following environment variable:

```
JAVA_TOOL_OPTIONS=-agentlib:jdwp=transport=dt_socket,address=8000,server=y,suspend=n
```

Debug Python app with VSCode

Debugging local applications with VSCode is simple. In the background, VSCode uses a python module to make the interface between the python interpreter and the IDE: [debugpy](#). This module is embedded in VSCode installation and is also available as a PIP package:



```
pip install debugpy
```

A screenshot of a terminal window with a dark background. The title bar is dark grey with three small icons: a minus sign, a square, and a red circle with a cross. The main area of the terminal shows the command "pip install debugpy" in white text.

The python application will open a debug port and the IDE will connect to this port.

Debug Python – Attach to running process

To attach to a running python app you need to:

- run the app with the `debugpy` module
- Configure VSCode to connect to the debug port

Run the app with debug features:



Then use the following VSCode launch config:

```
{
  "name": "Python: Attacher",
  "type": "python",
  "request": "attach",
  "connect": {
    "host": "127.0.0.1",
    "port": 5678
  },
  "pathMappings": [
    {
      "localRoot": "${workspaceFolder}",
      "remoteRoot": "."
    }
  ]
}
```

Debug Python – Debug Container

To be able to debug a container, you will need to:

- Load `debugpy` module in container:
 - install module
 - launch app with module
- Configure access from VSCode to container debug port
 - configure `debugpy` to listen on all interfaces instead of `localhost`
 - configure VSCode to access container

Debug Java app with VSCode

The following VSCode configuration connects to a java process running on localhost:8000:

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "type": "java",  
      "name": "Attach to Remote Program",  
      "request": "attach",  
      "hostName": "localhost",  
      "port": "8000"  
    }  
  ]  
}
```

Lab 6.1: Debug with VSCode



Objective

- Run application in container and run debugger outside container on workstation.

Steps

- Adapt `Dockerfile` to allow debug of app (load library, change compilation flags, ...)
- Rebuild your image and create a new container
- Connect VSCode Debugger to your container
- Debug step by step a call to `/buy/georchestra`

Questions

- Why not deploy this new container to prod?
 - Because the image is bigger
 - Debug libs may introduce security issues
 - Debug libs may slow down app
- Can I debug a container without rebuilding?
 - Yes
 - No
 - Depends on language

Labs solutions

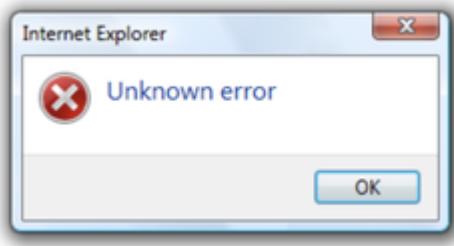
You can find an implementation of labs [here](#).

Develop inside container with VSCode

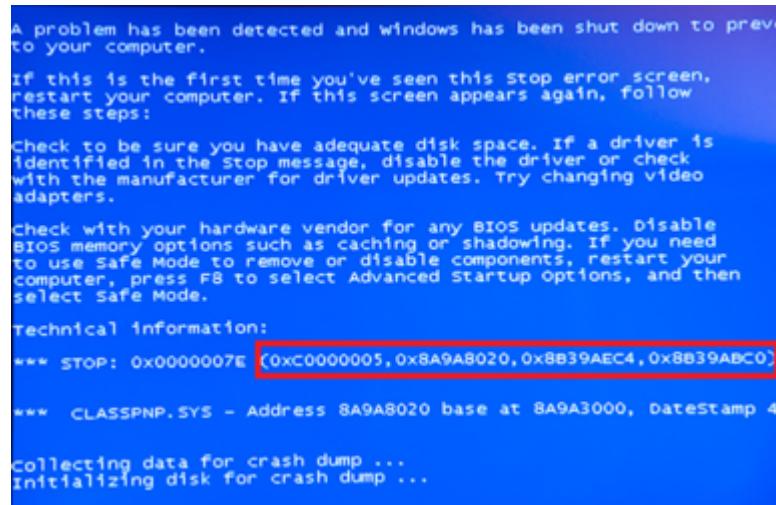
- Run a VSCode server
- Mount workspace
- Open port to connect IDE
- Use dedicate container image, include VSCode headless

Logging Bad practices (1/2)

- Avoid logs without information



- Avoid giving technical codes humans can't understand



- Avoid insulting the user :-)



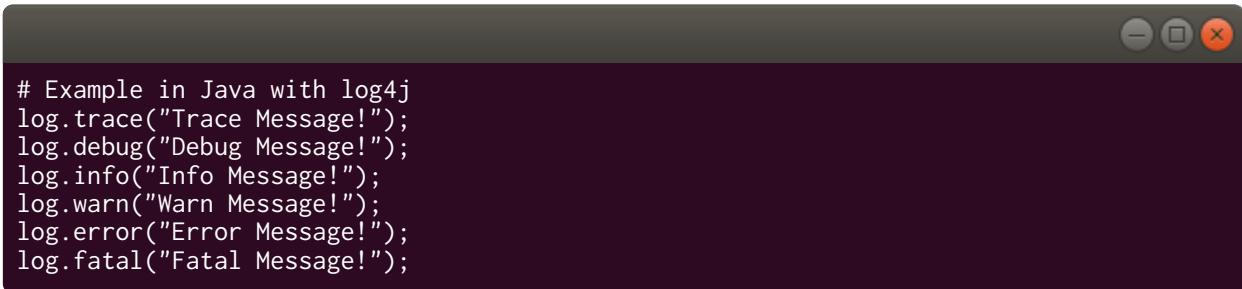
Bad practices (2/2)

- Avoid giving too much information (example : infamous Java Stack Trace)

```
2021-02-04 10:19:19.218 ERROR 1 --- [qtp250075633-23] o.s.b.w.servlet.support.ErrorPage
org.json.simple.parser.ParseException: null
    at org.json.simple.parser.Yylex.yylex(Yylex.java:610) ~[json-simple-1.1.1.jar:na]
    at org.json.simple.parser.JSONParser.nextToken(JSONParser.java:269) ~[json-simple-1
    at org.json.simple.parser.JSONParser.parse(JSONParser.java:118) ~[json-simple-1.1.1
    at org.json.simple.parser.JSONParser.parse(JSONParser.java:92) ~[json-simple-1.1.1
    at com.camptocamp.containerscourseapp.ProductsController.readDataJson(ProductsContr
    at com.camptocamp.containerscourseapp.ProductsController.readProductList(ProductsCo
    at com.camptocamp.containerscourseapp.ProductsController.index(ProductsController.j
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[na:1.8.0_282]
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62) ~[
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav
    at java.lang.reflect.Method.invoke(Method.java:498) ~[na:1.8.0_282]
    at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(Invocabl
    at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(I
    at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMet
    at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapt
    at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapt
    at org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(A
    at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.j
    at org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.ja
    at org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServle
    at org.springframework.web.servlet.FrameworkServlet doGet(FrameworkServlet.java:898
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:687) ~[servlet-api-3.1.j
    at org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:8
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:790) ~[servlet-api-3.1.j
    at org.eclipse.jetty.servlet.ServletHolder.handle(ServletHolder.java:791) ~[na:na]
    at org.eclipse.jetty.servlet.ServletHandler$ChainEnd.doFilter(ServletHandler.java:1
    at org.eclipse.jetty.websocket.server.WebSocketUpgradeFilter.doFilter(WebSocketUpgr
    at org.eclipse.jetty.servlet.FilterHolder.doFilter(FilterHolder.java:193) [jetty-se
    at org.eclipse.jetty.servlet.ServletHandler$Chain.doFilter(ServletHandler.java:1601
    at org.springframework.web.filter.RequestContextFilter.doFilterInternal(RequestCont
    at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilte
    at org.eclipse.jetty.servlet.FilterHolder.doFilter(FilterHolder.java:193) [jetty-se
<...>
```

Use log levels

- Traditionally : TRACE, DEBUG, INFO, WARN, ERROR, FATAL
- Logs request are attached to a specific Log Level



```
# Example in Java with log4j
log.trace("Trace Message!");
log.debug("Debug Message!");
log.info("Info Message!");
log.warn("Warn Message!");
log.error("Error Message!");
log.fatal("Fatal Message!");
```

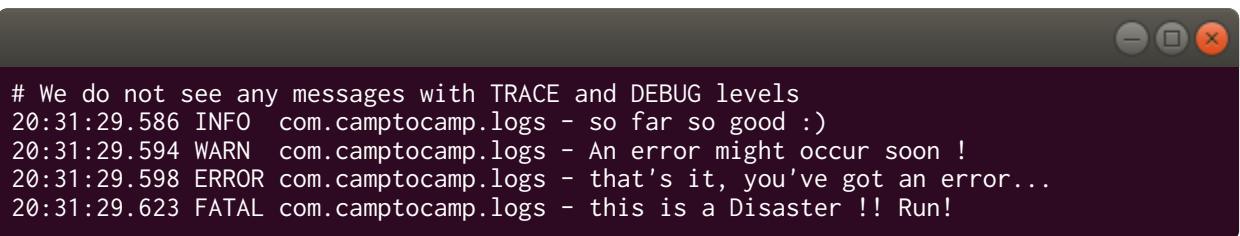
A screenshot of a terminal window with a dark background. The window has standard Linux-style window controls at the top right. Inside the window, there is a single line of Java code demonstrating the use of log levels. The code uses the log object to call trace, debug, info, warn, error, and fatal methods, each passing a string message as an argument.

Configure Log Level

- Application can filter logs displayed

```
<!-- Example of configuration using SLF4J/Logback -->
<configuration>
    <..>
    <root level="info">
        <appender-ref ref="console"/>
    </root>
</configuration>
```

- Level set to info, logs at selected level and above are displayed



```
# We do not see any messages with TRACE and DEBUG levels
20:31:29.586 INFO com.camptocamp.logs - so far so good :)
20:31:29.594 WARN com.camptocamp.logs - An error might occur soon !
20:31:29.598 ERROR com.camptocamp.logs - that's it, you've got an error...
20:31:29.623 FATAL com.camptocamp.logs - this is a Disaster !! Run!
```

Format information in a readable way

- Add useful extra information to each log entry
- Organize data to simplify reading and parsing

```
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss} [%thread] %-5level %logger{10} - %msg%n%rEx{3}</pattern>
    </encoder>
  </appender>
</configuration>
```

- In Java, shorten (and even reverse) infamous Stack traces

```
15:53:55 [qtp250075633-12] DEBUG c.c.c.ProductsController - Reading products from file:
15:53:55 [qtp250075633-12] ERROR o.s.b.w.s.ErrorPageFilter - Forwarding to error page
org.json.simple.parser.ParseException: null
    at org.json.simple.parser.Yylex.yylex(Yylex.java:610)
    at org.json.simple.parser.JSONParser.nextToken(JSONParser.java:269)
15:53:56 [qtp250075633-12] DEBUG c.c.c.ProductsController - Reading products from file:
```

Add metadata and extra informations to the logs

Logs can be formatted in JSON. This help tools like ElasticSearch to index and to perform advanced search.

Logs do not only contain a text message but additional informations:

- Source IP address
- User Agent
- HTTP Referer
- User

Filter logs for one customer.

Checkpoint: Application Development

- The following languages can be debugged without rebuilding the container image
 - Python
 - Go
 - Java
 - PHP

- Following languages require rebuild for smoke testing in container
 - Python
 - Go
 - Java
 - PHP

PUBLISHING OCI IMAGES

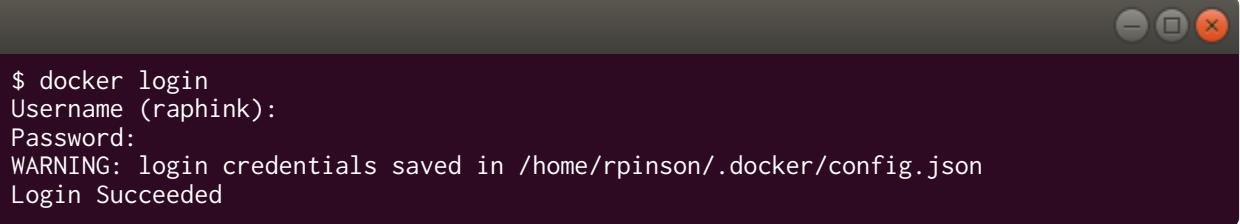
Lesson 7: Publishing Images

Objectives

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

- log in to a registry
- push images to a registry

Log in to DockerHub

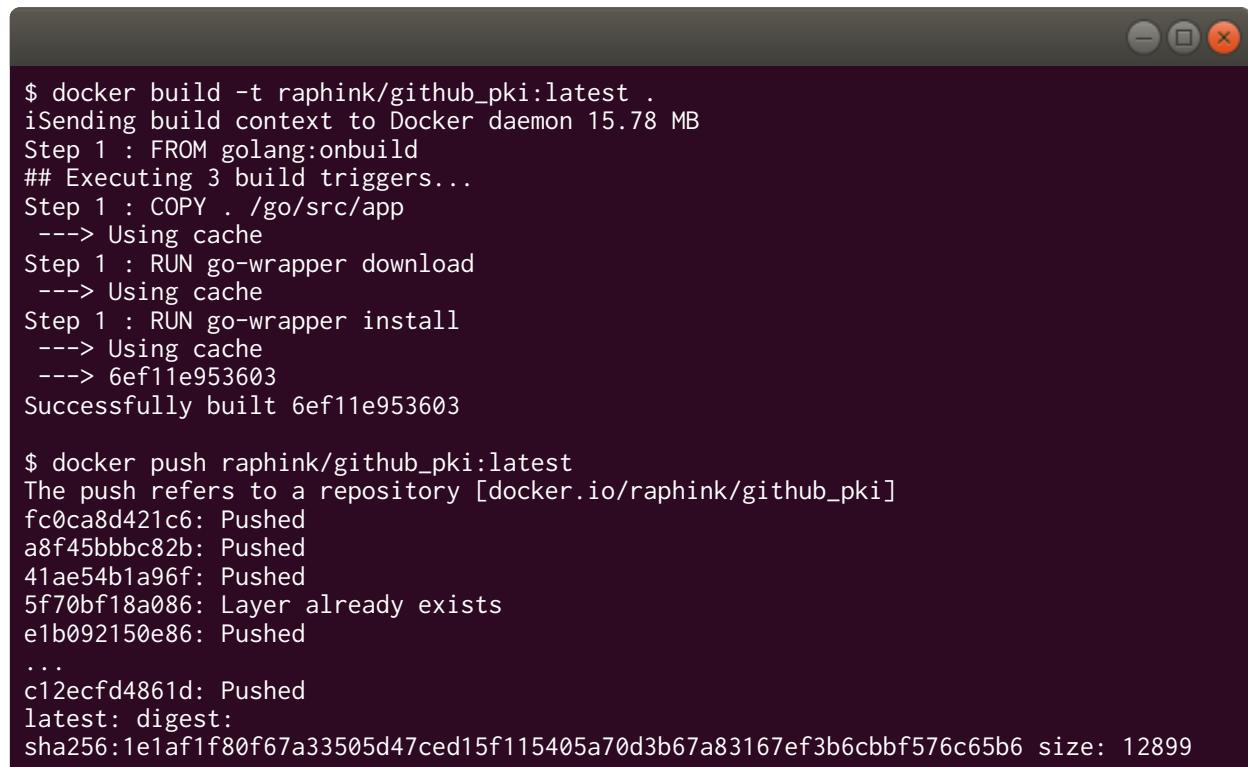


```
$ docker login
Username (raphink):
Password:
WARNING: login credentials saved in /home/rpinson/.docker/config.json
Login Succeeded
```

A screenshot of a terminal window with a dark background. The window has standard Linux-style window controls at the top right. The terminal displays the command '\$ docker login' followed by a password prompt. It then shows a warning message about saving credentials and finally 'Login Succeeded'.

Push image to DockerHub

- tag image as <user>/<repository>[:<tag>]
- push image to registry using the tag

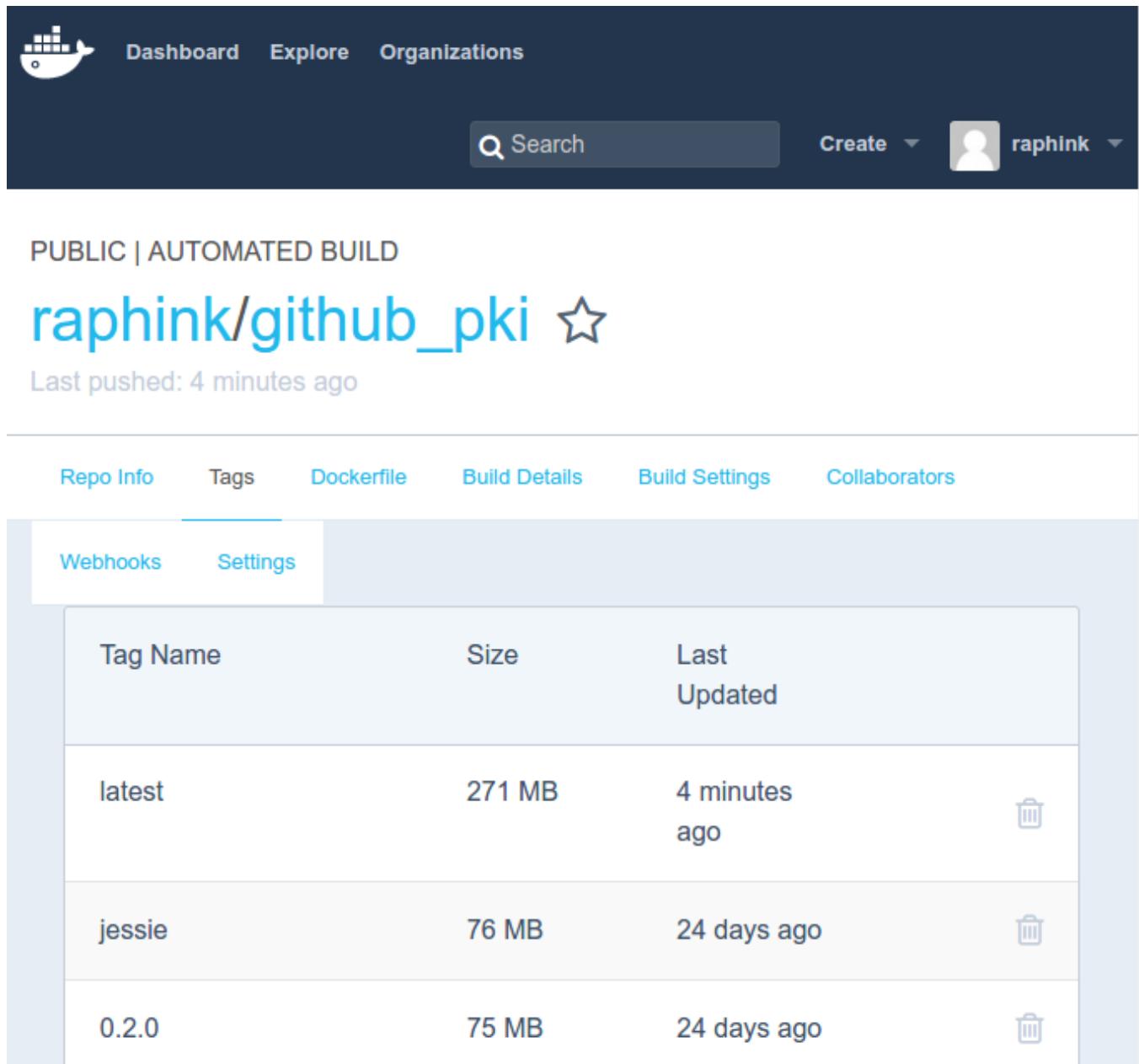


```
$ docker build -t raphink/github_pki:latest .
iSening build context to Docker daemon 15.78 MB
Step 1 : FROM golang:onbuild
## Executing 3 build triggers...
Step 1 : COPY . /go/src/app
    ---> Using cache
Step 1 : RUN go-wrapper download
    ---> Using cache
Step 1 : RUN go-wrapper install
    ---> Using cache
    ---> 6ef11e953603
Successfully built 6ef11e953603

$ docker push raphink/github_pki:latest
The push refers to a repository [docker.io/raphink/github_pki]
fc0ca8d421c6: Pushed
a8f45bbbc82b: Pushed
41ae54b1a96f: Pushed
5f70bf18a086: Layer already exists
e1b092150e86: Pushed
...
c12ecfd4861d: Pushed
latest: digest:
sha256:1e1af1f80f67a33505d47ced15f115405a70d3b67a83167ef3b6cbbf576c65b6 size: 12899
```

Images on DockerHub

DockerHub allows to access all pushed tags

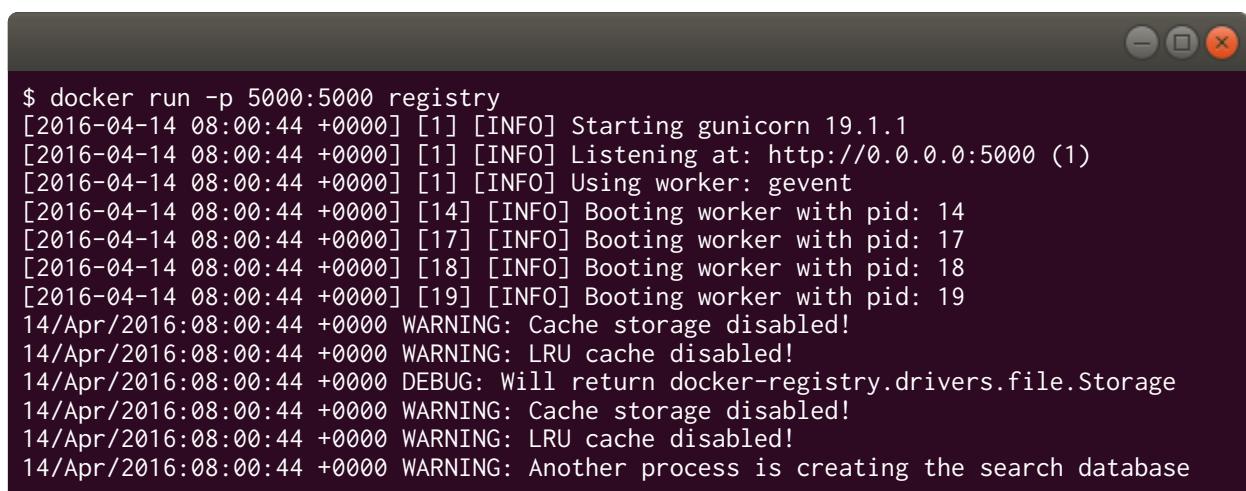


The screenshot shows the DockerHub interface for the repository `raphink/github_pki`. The repository has three tags: `latest`, `jessie`, and `0.2.0`. The `latest` tag was pushed 4 minutes ago and is 271 MB in size. The `jessie` tag is 76 MB in size and was pushed 24 days ago. The `0.2.0` tag is 75 MB in size and was also pushed 24 days ago. Each tag row includes a trash can icon for deletion.

Tag Name	Size	Last Updated	Action
latest	271 MB	4 minutes ago	
jessie	76 MB	24 days ago	
0.2.0	75 MB	24 days ago	

Setting up a registry

- use the `registry` image
- no web interface provided
- authentication can be provided with `htpasswd`, `silly` or `token`
- beware of persisting data (can be stored on object storage)
- See [registry configuration](#)



A terminal window showing the output of a Docker command. The command is \$ docker run -p 5000:5000 registry. The log output shows the registry starting gunicorn 19.1.1, listening on http://0.0.0.0:5000, and booting workers with pids 14, 17, 18, and 19. It also shows multiple WARNING messages about cache storage being disabled and DEBUG messages about storage drivers.

```
$ docker run -p 5000:5000 registry
[2016-04-14 08:00:44 +0000] [1] [INFO] Starting gunicorn 19.1.1
[2016-04-14 08:00:44 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
[2016-04-14 08:00:44 +0000] [1] [INFO] Using worker: gevent
[2016-04-14 08:00:44 +0000] [14] [INFO] Booting worker with pid: 14
[2016-04-14 08:00:44 +0000] [17] [INFO] Booting worker with pid: 17
[2016-04-14 08:00:44 +0000] [18] [INFO] Booting worker with pid: 18
[2016-04-14 08:00:44 +0000] [19] [INFO] Booting worker with pid: 19
14/Apr/2016:08:00:44 +0000 WARNING: Cache storage disabled!
14/Apr/2016:08:00:44 +0000 WARNING: LRU cache disabled!
14/Apr/2016:08:00:44 +0000 DEBUG: Will return docker-registry.drivers.file.Storage
14/Apr/2016:08:00:44 +0000 WARNING: Cache storage disabled!
14/Apr/2016:08:00:44 +0000 WARNING: LRU cache disabled!
14/Apr/2016:08:00:44 +0000 WARNING: Another process is creating the search database
```

Pushing to a private registry

- set the registry URL in the tag

```
$ docker tag raphink/github_pki:latest localhost:5000/raphink/github_pki:latest
$ docker push localhost:5000/raphink/github_pki:latest
The push refers to a repository [localhost:5000/raphink/github_pki]
fc0ca8d421c6: Image successfully pushed
a8f45bbbc82b: Image successfully pushed
41ae54b1a96f: Image successfully pushed
5f70bf18a086: Image successfully pushed
e1b092150e86: Image successfully pushed
21ac53bc7cd6: Image successfully pushed
6e2e798f8998: Image successfully pushed
f9f3fb66a490: Image successfully pushed
0b2fe2c6ef6b: Image successfully pushed
591569fa6c34: Image successfully pushed
998608e2fcfd4: Image successfully pushed
c12ecfd4861d: Image successfully pushed
Pushing tag for rev [6ef11e953603] on {http://localhost:5000/v1/repositories/raphink/
github_pki/tags/latest}
```

Registry Vocabulary

- a registry hosts repositories
- a repository hosts tags for an image

Generally:

```
<registry>/<user>/<repository>:<tag>
```



Push/Pull troubleshooting

- On docker hub:
 - pushing to a non-existing repository will create a new private one
 - there is *authentication*: private repository are not accessible
 - there is *authorization*: you need explicit rights to access a private repository

- The default tag for an image is `latest`:
 - `pull` without a tag downloads the `latest` tag
 - `push` without tag pushes *all* tags

Exercise 7.1: push to a registry



Objective

- Push your image to a registry

Steps

- Launch a docker registry on your workstation
- Tag your image
- Push your image to your local registry or DockerHub
- Delete image locally
- Run `docker pull` to retrieve the image from the registry

Questions

- Formatting tags properly is important to publish images
 - Yes
 - No
- All layers are always pushed when pushing an image
 - Yes
 - No

Checkpoint: Publishing Images

When publishing images to a registry:

- Setting up a private registry is trivial
 - Yes
 - No
- A repository may contain
 - Multiple tags for an image
 - Multiple images for a tag
 - Multiple users
 - Multiple registries
- A unique tag may point to different images
 - Yes
 - No

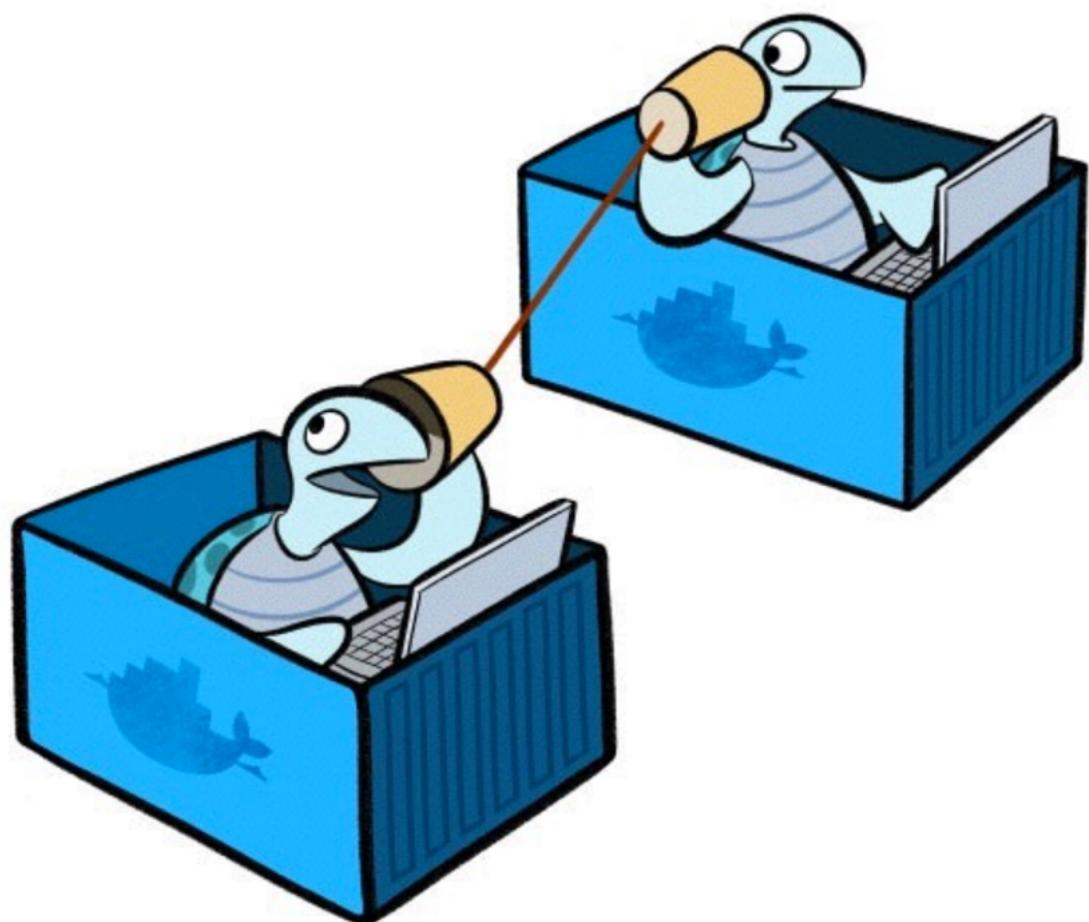
PORts & HOSTs

Lesson 8: Ports & hosts

Objectives

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

- communicate between containers
- publish container ports on the host machine
- connect a container to a remote service



Default network environment

By default:

- containers do not resolve each other's names
- container ports are not published on the Docker host
- each container has a dedicated network interface

Link containers with /etc/hosts

- make container know each other
- uses /etc/hosts entries

```
$ docker run -ti --name foo busybox sh
## New shell
$ docker run busybox ping -c 1 foo
ping: bad address 'foo'

$ docker run --link foo busybox ping -c 1 foo
PING foo (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.135 ms
--- foo ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.138/0.138/0.138 ms

$ docker run --link foo busybox grep foo /etc/hosts
172.17.0.3      foo 820aa7b850ee
```

Notes:

- creating networks in Docker 1.10+ provides native DNS resolution

Expose container ports

- specify which ports are exposed by a container
- in Dockerfile

```
EXPOSE 80
```

- when creating/running

```
$ docker create --expose 80 busybox
$ docker run --expose 80 busybox
```

Inter-containers communication

Let containers communicate!

```
$ docker run -d -e FOO=bar --name hello tutum/hello-world
21812b41e8b3a8168ef37e4426a51a187cbc8f9aa09aa37234ce8708bba5d17a

$ docker run --link hello curlimages/curl -s http://hello
<html>
<head>
    <title>Hello world!</title>
</head>
<body>
    
    <h1>Hello world!</h1>
    <h3>My hostname is 21812b41e8b3</h3>      </body>
</html>
```

Linking exports environment

Exported environments are prefixed with the link name:

```
$ docker run --link hello curlimages/curl env  
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
HOSTNAME=92d0a9d90d07  
HELLO_PORT=tcp://172.17.0.6:80  
HELLO_PORT_80_TCP=tcp://172.17.0.6:80  
HELLO_PORT_80_TCP_ADDR=172.17.0.6  
HELLO_PORT_80_TCP_PORT=80  
HELLO_PORT_80_TCP_PROTO=tcp  
HELLO_NAME=/gloomy_darwin/hello  
HELLO_ENV_FOO=bar  
HOME=/
```

Exercise 8.1: inter-containers communication



Objective

- Make two containers communicate over the Docker network

Steps

- Run a first container with exposed ports and a name
- Run a second container linking to the first one and access the exposed port

Questions

- How can this simplify application configurations in containers?
 - Service ports can be set statically
 - Service IPs can be set statically
 - Service DNS names can be set statically
 - Database names can be set statically

Ports are not published by default

```
$ docker run -d --expose 80 --name hello tutum/hello-world  
21812b41e8b3a8168ef37e4426a51a187cbc8f9aa09aa37234ce8708bba5d17a  
  
$ docker ps  
CONTAINER ID        IMAGE               COMMAND             CREATED            NAMES  
c65130072483        tutum/hello-world   "/bin/sh -c 'php-fpm'"   50 seconds ago   hello  
                      STATUS              PORTS             NAMES  
                      Up 49 seconds      80/tcp           hello  
  
$ curl http://localhost  
curl: (7) Failed to connect to localhost port 80: Connection refused
```

Publishing ports

- publishing a port binds it on the host
- `--publish-all` publishes all exposed ports to random host ports

```
$ docker rm -f hello && docker run -d --name hello --publish-all tutum/hello-world
hello
c65130072483373fd3fc3e3148244d1fdaeabbf480e3457a27d288717a21e208

$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                 NAMES
c65130072483        tutum/hello-world   "/bin/sh -c 'php-fpm '"   50 seconds ago    Up 49 seconds      0.0.0.0:32770->80/tcp   hello

$ curl -s http://localhost:32770
<html>
<head>
  <title>Hello world!</title>
</head>
<body>
  
  <h1>Hello world!</h1>
  <h3>My hostname is c65130072483</h3>
</body>
</html>
```

Publishing specific ports

```
--publish <port>
```

Exposes *and* publishes one port to a random host port

```
$ docker rm -f hello && docker run -d --name hello --publish 80 tutum/hello-world
hello
0bb4aa75c61bb853123003d717d8d865fc5fe633a129c2fea928c4a64a0e409

$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
0bb4aa75c61b        tutum/hello-world   "/bin/sh -c 'php-fpm'"   18 seconds ago   Up 18 seconds      hello

$ curl -s http://localhost:32771
<html>
<head>
  <title>Hello world!</title>
</head>
<body>
  
  <h1>Hello world!</h1>
  <h3>My hostname is 0bb4aa75c61b</h3>
</body>
</html>
```

Publishing to specific ports

```
--publish [host_ip:] [host_port:] <container_port>
```

Exposees and publishes one port to a specific host port (provided it is available)

```
$ docker rm -f hello && docker run -d --name hello --publish 8080:80 tutum/hello-world
hello
790edd1dbed1a2612a9f8809f107b33ac09e827edd1079391facb2fb57709b79

$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              NAMES
790edd1dbed1        tutum/hello-world   "/bin/sh -c 'php-fpm'"   20 seconds ago    Up 20 seconds     hello

$ curl -s http://localhost:8080
<html>
<head>
  <title>Hello world!</title>
</head>
<body>
  
  <h1>Hello world!</h1>
  <h3>My hostname is 790edd1dbed1</h3>
</body>
</html>
```

Exercise 8.2: publish container ports



Objective

- Publish container services on the host

Steps

- Run containers and publish their services on the host, using dynamic and fixed ports

Questions

- What do you need to watch when publishing ports?
 - Publishing reserved ports
 - Running containers on the same node
 - Running containers in the same Docker network

Linking to a remote service

- `--add-host <host>:<ip>` inserts a host in `/etc/hosts` in a container

```
$ host ianahazip.com
ianahazip.com has address 64.182.208.182
ianahazip.com has address 64.182.208.181
ianahazip.com has IPv6 address 2604:7780:200:305:f816:3eff:fee0:d985
ianahazip.com has IPv6 address 2604:7780:200:305:f816:3eff:fe5b:d7ce

$ docker run --add-host myip:64.182.208.182 curlimages/curl -s http://myip
81.18.187.187
```

Checkpoint: Ports & Hosts

When dealing with ports:

- Containers know each other by name by default
 - Yes
 - No
- Exposing a port can be done
 - in the Dockerfile
 - when creating a container
 - when starting/restarting a container
- Publishing a port can be done
 - in the Dockerfile
 - when creating a container
 - when starting/restarting a container

DOCKER NETWORKS

Lesson 9: Network

Objectives

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

- create and use networks
- associate a container to a network
- use DNS round-robin between containers

Create a network

- default network is `bridge` (`docker0` host interface)

```
$ docker network ls
NETWORK ID      NAME      DRIVER
00ad26f809ea   bridge    bridge
da10bf136cfa   host      host
62c56d78b44b   none     null

$ docker network create foo
fd0937f4e569f58296e8c17a7e6a30e10040a1d0943467222bba073c13ae5390

$ docker network ls
NETWORK ID      NAME      DRIVER
00ad26f809ea   bridge    bridge
fd0937f4e569   foo       bridge
da10bf136cfa   host      host
62c56d78b44b   none     null
```

Inspect a network

```
$ docker network inspect foo
[
    {
        "Name": "foo",
        "Id": "fd0937f4e569f58296e8c17a7e6a30e10040a1d0943467222bba073c13ae5390",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Internal": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]

$ docker network inspect -f "{{.IPAM}}" bridge
{default map[] [{172.17.0.0/16 172.17.0.1 map[]}]}
```

Run a container in a network

```
$ docker run --net foo busybox ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
156: eth0@if157: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe12:2/64 scope link tentative
        valid_lft forever preferred_lft forever
```

Notes:

- `docker network connect <NETWORK> <CONTAINER>` lets you add a container to an existing network

Networks are filtered

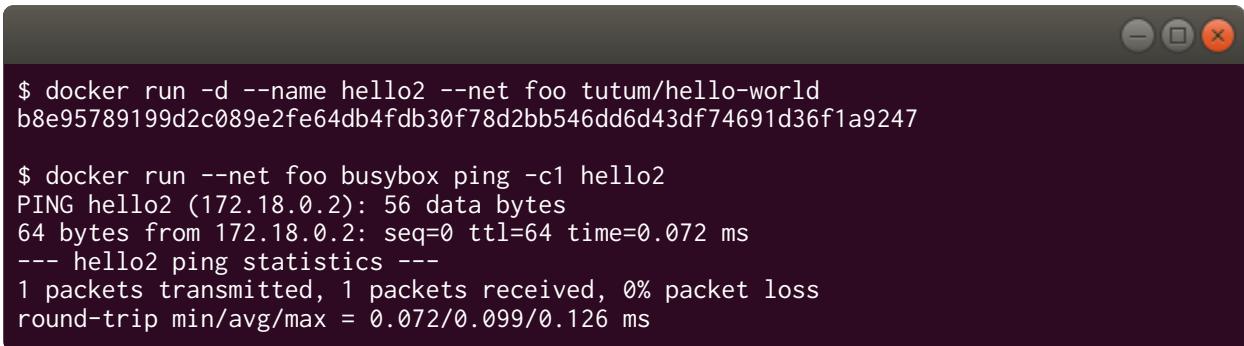
- containers can not communicate between networks

```
$ docker run --link hello busybox ping -c1 hello
PING hello (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.139 ms
--- hello ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.139/0.139/0.139 ms

$ docker run --link hello --net foo busybox ping -c1 hello
ping: bad address 'hello'
```

Use DNS resolution

- provides an automatic linking feature (no `/etc/hosts` edition)
- Docker 1.10+ feature



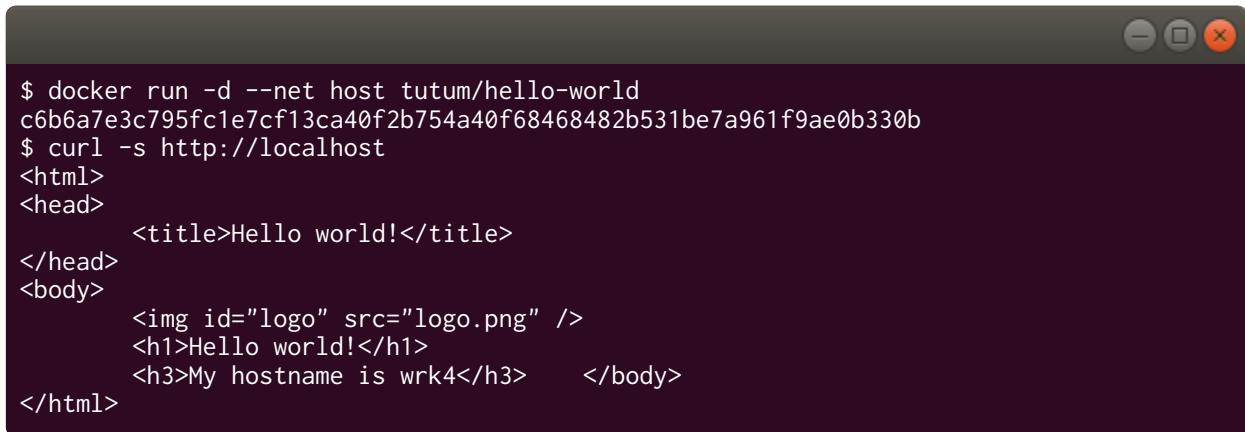
```
$ docker run -d --name hello2 --net foo tutum/hello-world
b8e95789199d2c089e2fe64db4fdb30f78d2bb546dd6d43df74691d36f1a9247

$ docker run --net foo busybox ping -c1 hello2
PING hello2 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.072 ms
--- hello2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.072/0.099/0.126 ms
```

- the `bridge` network has no DNS resolution

The host network

- the host network is shared with the host (exposed ports are published)



```
$ docker run -d --net host tutum/hello-world
c6b6a7e3c795fc1e7cf13ca40f2b754a40f68468482b531be7a961f9ae0b330b
$ curl -s http://localhost
<html>
<head>
    <title>Hello world!</title>
</head>
<body>
    
    <h1>Hello world!</h1>
    <h3>My hostname is wrk4</h3>
</body>
</html>
```

Exercise 9.1: use networks



Objective

- Create a network and run containers in it

Steps

- Create a new Docker network
- Run two containers in it and link them using DNS resolution

Questions

- Does this work with containers on different Docker nodes?
 - Yes
 - No

Checkpoint: Networks

When using Docker networks

- Containers know each other by name
 - Always
 - When linking them
 - When running on the same declared network
- Containers can be assigned to multiple networks
 - Yes
 - No
- It is usually simpler to just use the host network
 - Yes
 - No
 - YOLO

DOCKER VOLUMES

Lesson 10: Volumes

Objectives

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

- understand the different ways to bind volumes to containers
- persist data to disk
- share data between containers

Use volumes in containers

- use `docker run -v`
- volumes are *not* suppressed when containers are removed
- using volumes improves performance (local FS vs AUFS/overlay/etc. layer data)
- 3 different syntaxes (since Docker 1.8)

```
$ docker run -v /var/cache busybox
$ docker run -v /host/path:/container/path busybox
$ docker run -v mydata:/data busybox
```

Declare volume in container

- declare `/var/cache` as a separated volume in the container (external to the layer stack, stored on host FS)
- equivalent to using a `VOLUME` directive in `Dockerfile`
- the content from `/var/cache` in the *image* is copied to `/var/cache` in the container to initialize the data
- using `--rm` destroys the volume when the container is destroyed

```
$ docker run -v /var/cache busybox
```

Exercise 10.1: Declare volume in container



Objective

- Run a container with a declared volume

Steps

- Run an image and declare a volume
- Inspect the container to find the volume that was created
- Add a file to the volume
- Restart the container, check that the data is still there
- Remove and recreate the container, check that the data is gone
- Create a second container, check that the data is not shared with the first container

Questions

- What filesystem type can be used with volumes?
 - ext4
 - ZFS
 - glusterfs
 - any supported by your OS
- What could this method be used for?
 - Persisting data
 - Cache
 - Shared data between containers
 - Configuration
 - Secrets

Mount local filesystem in container

- mount bind `/host/path` from host filesystem as `/container/path` in container
- no way to initialize the data (data from filesystem is used, even if empty)
- not easily portable (multi-host setups)
- beware of containers mounting the same directory!

```
$ docker run -v /host/path:/container/path busybox
```

Notes:

- files can be mounted as well, but they will not be kept in sync, instead they will be copied to the container:

```
$ docker run -d \
-v $PWD/index.html:/usr/share/nginx/html/index.html \
nginx
```

Exercise 10.2: Mount local directory in container



Objective

- Run a container with a mounted directory from host

Steps

- Run an image and mount-bind a directory from the host
- Check that the data in the container is the one from the host
- Change data in the container and check that it is modified on the host
- Remove and recreate the container and check that the data is still there
- Create a second container, check that the data is shared with the first container

Questions

- What happens if you remove a directory mounted in a container?
 - Nothing
 - It unmounts it
 - It deletes it on the host node
 - You can't do that
- What could this method be used for?
 - Persisting data
 - Cache
 - Shared data between containers
 - Configuration
 - Secrets

Notes:

- What would happen if you run the following?

```
$ docker run -v /:/target busybox cat /target/etc/shadow
```

- What are the implications in terms of security?

Mount named volume in container

- available since Docker 1.8
- mount bind the `mydata` named volume (located in `/var/lib/docker/volumes/mydata/_data`) as `/data` in container
- data initialized from image (since Docker 1.10)
- namespace is not private to the container, beware of containers mounting the same named volume!

```
$ docker run -v mydata:/data busybox
```

Exercise 10.3: Mount named volume in container



Objective

- Run a container with a named volume

Steps

- Run an image with a named volume
- Modify the data in the container
- Stop and start the container, check that the data is still there
- Remove and recreate the container, check that the data is still there
- Create a second container, check that the data is shared with the first container

Questions

- How could you tell if a volume is named?
 - It has a special metadata
 - Its name is not 64 characters long
- What could this method be used for?
 - Persisting data
 - Cache
 - Shared data between containers
 - Configuration
 - Secrets

Persist data to disk

- both mount-bind and named volume methods allow to persist data to disk
- beware of data shared between containers
- enables (but does not automatically perform) backup
- not simple to implement on multi-host platforms

Sharing volumes between containers

- both mount-bind and named volume methods allow to share volumes

```
$ docker run -ti --name mycont -v mydata:/data busybox sh
## New shell
$ docker run -v mydata:/data:ro busybox mount | grep data
/dev/sda1 on /data type ext4 (ro,relatime,errors=remount-ro,data=ordered)
```

Sharing volumes between containers

- declared volumes from a first container in a second container

```
$ docker run -ti --name mycont -v /data busybox sh
## New shell
$ docker run --volumes-from=mycont busybox mount | grep data
/dev/sda1 on /data type ext4 (rw,relatime,errors=remount-ro,data=ordered)
```

- A way to inject configuration in upstream container
- Not available in orchestrator:
 - Deprecated on docker-compose v3
 - Available in Rancher 1.6...
 - ... but Rancher 1.6 provider secrets

Notes:

- note: `:ro` can be used to mount-bind a volume in read-only mode

Exercise 10.4: Share a volume between containers



Objective:

- Run two containers with shared data

Steps

- Run a container with declared volumes (in the Dockerfile or on the command line)
- Run a second container with `--volumes-from`
- Check that the volume is shared
- Stop and remove both containers
- Share data between two containers using a named volume

Questions

- How would you set up a producer/consumer system?
 - Use a `:ro` flag on the consumer
 - Ensure both containers run on the same node
 - Use glusterfs for `/var/lib/docker/volumes`
 - Use an API instead of a volume

Secret and ConfigMap

Volume can be used to provide secret or configuration

- Kubernetes have ConfigMap: collection of key -> value
 - environment variables
 - configuration files
- Kubernetes also have Secret
- Rancher 1.6 have secrets

Checkpoint: Volumes

When using volumes in containers:

- Volumes can be declared
 - in the Dockerfile
 - when a container is created
 - when a container is started/restarted
- Declaring a volume empties its content
 - Yes
 - No
- Volumes can be mounted from any path on the host node
 - Yes
 - No
- Volumes can be shared between containers on multiple nodes
 - Yes
 - No
- What do you need to watch for when sharing volumes?
 - User IDs
 - Data size
 - Running containers on the same node
 - Running containers in the same Docker network

DOCKER-COMPOSE

Lesson 11: Docker-Compose

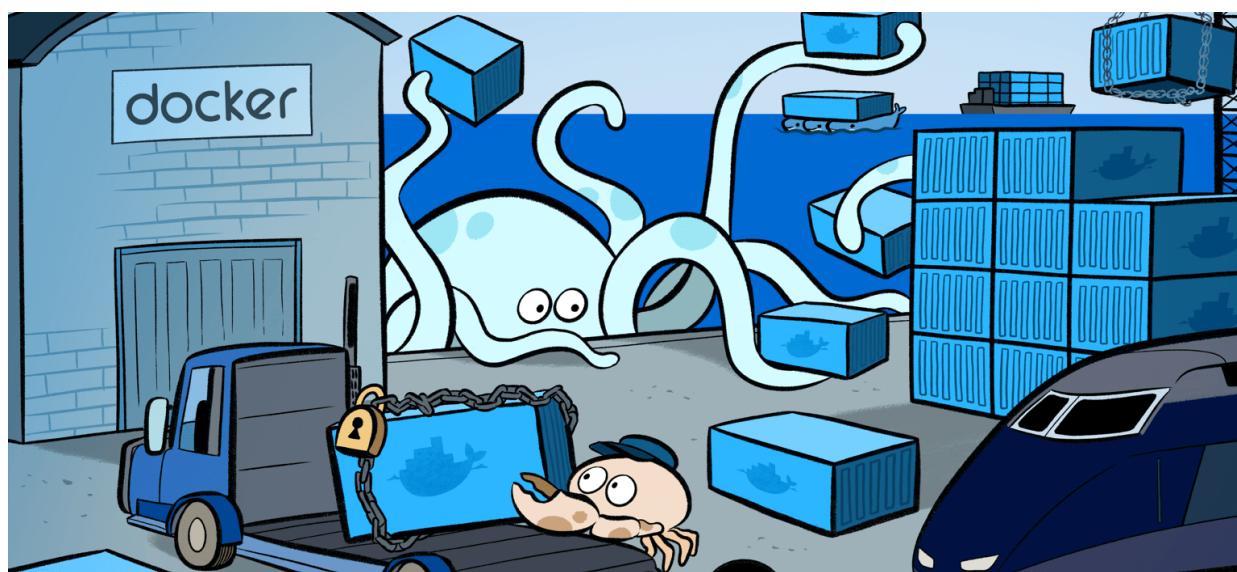
Objectives

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

- write a `docker-compose.yml` file
- start, stop and remove a composition
- view the container logs

Container orchestration

- multiple containers
- interconnected (links)
- deployed from one source
- and versionned



Docker-compose

- formerly known as `fig`
- YAML-based format

```
---
```

```
web:
  build: .
  ports:
    - "5000:5000"
  volumes:
    - .:/code
  links:
    - redis
redis:
  image: redis
```

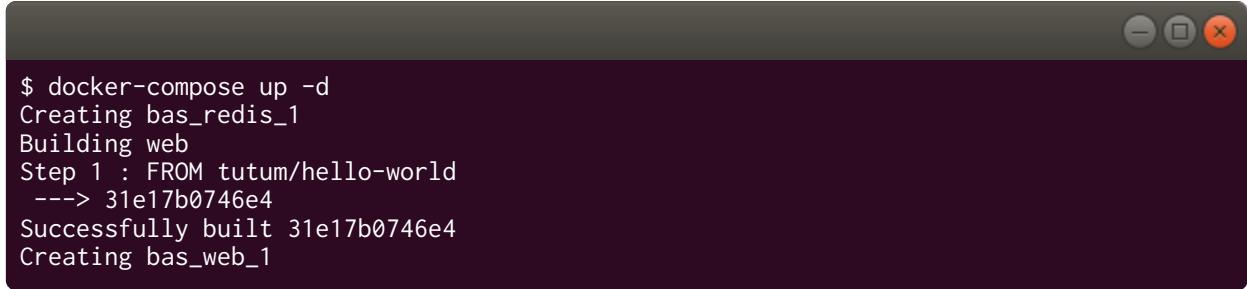
Install Compose

■ On Linux systems

```
$ sudo curl -L https://github.com/docker/compose/releases/download/1.27.4/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose  
$ chmod +x /usr/local/bin/docker-compose
```

Docker-compose up

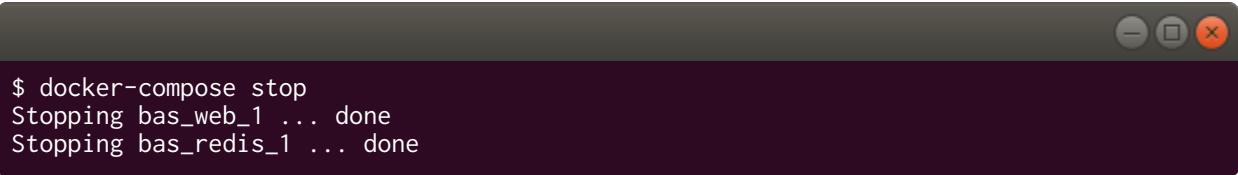
- docker-compose build + docker-compose start
- -d for detaching



```
$ docker-compose up -d
Creating bas_redis_1
Building web
Step 1 : FROM tutum/hello-world
--> 31e17b0746e4
Successfully built 31e17b0746e4
Creating bas_web_1
```

Docker-compose stop

- Stop all containers in the composition



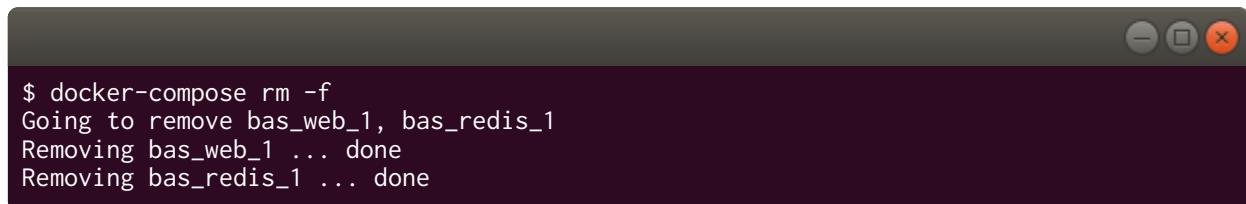
```
$ docker-compose stop
Stopping bas_web_1 ... done
Stopping bas_redis_1 ... done
```

A screenshot of a terminal window with a dark background and light-colored text. The window has standard OS X-style window controls at the top right. The terminal output shows the command '\$ docker-compose stop' followed by two lines of text indicating the stopping of services: 'Stopping bas_web_1 ... done' and 'Stopping bas_redis_1 ... done'. The text is in a monospaced font.

- `docker-compose kill` sends a `SIGKILL` instead of `SIGTERM` to services

Docker-compose rm

- Remove all containers in the composition
- `-f` stops and removes containers



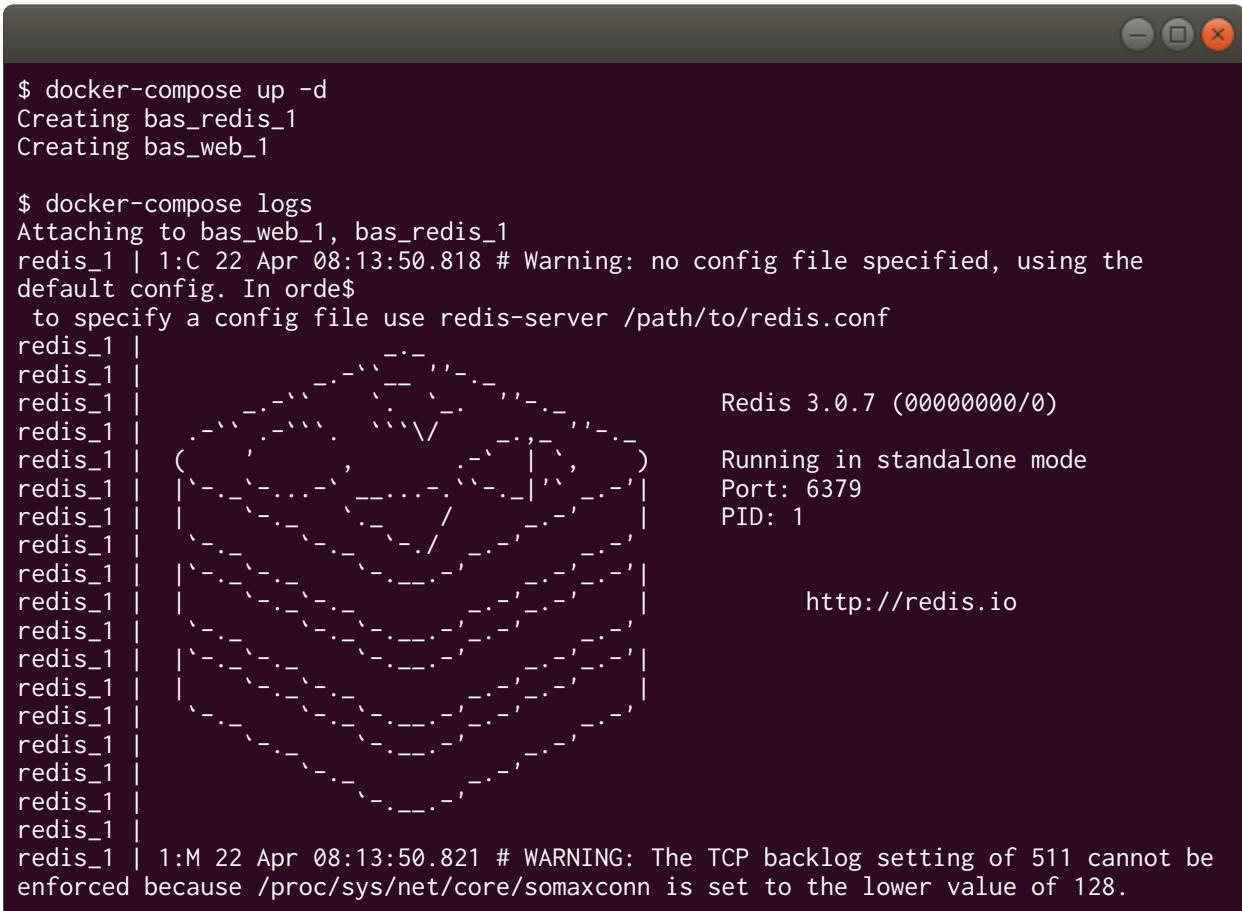
```
$ docker-compose rm -f
Going to remove bas_web_1, bas_redis_1
Removing bas_web_1 ... done
Removing bas_redis_1 ... done
```

- `docker-compose down` = `docker-compose stop` + `docker-compose rm`
- `docker-compose down --volumes --remove-orphans` removes:
 - Running container
 - volumes
 - volumes previously created (deleted from the manifest)

Docker-compose logs

- access logs of all containers in the composition

- useful when composition is detached

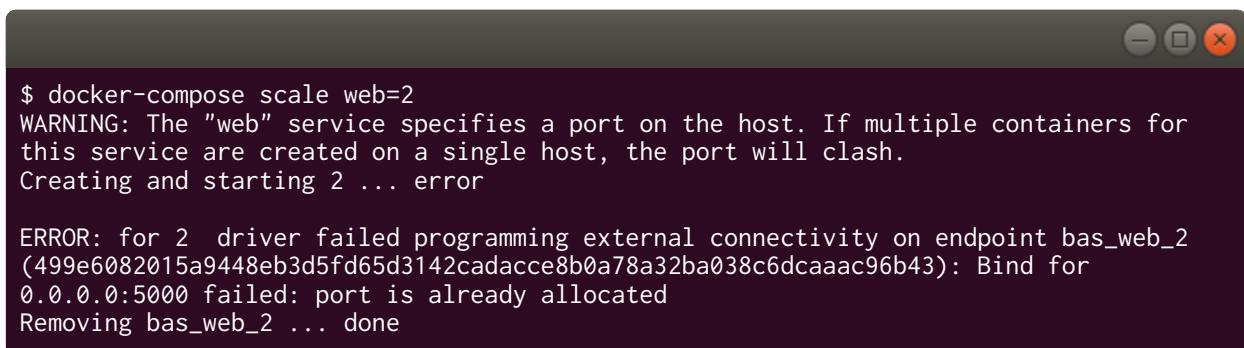


```
$ docker-compose up -d
Creating bas_redis_1
Creating bas_web_1

$ docker-compose logs
Attaching to bas_web_1, bas_redis_1
redis_1 | 1:C 22 Apr 08:13:50.818 # Warning: no config file specified, using the
redis_1 | default config. In orde$ to specify a config file use redis-server /path/to/redis.conf
redis_1 |                               Redis 3.0.7 (00000000/0)
redis_1 |                               Running in standalone mode
redis_1 |                               Port: 6379
redis_1 |                               PID: 1
redis_1 |                               http://redis.io
redis_1 |
redis_1 | 1:M 22 Apr 08:13:50.821 # WARNING: The TCP backlog setting of 511 cannot be
redis_1 | enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
```

Scale a service

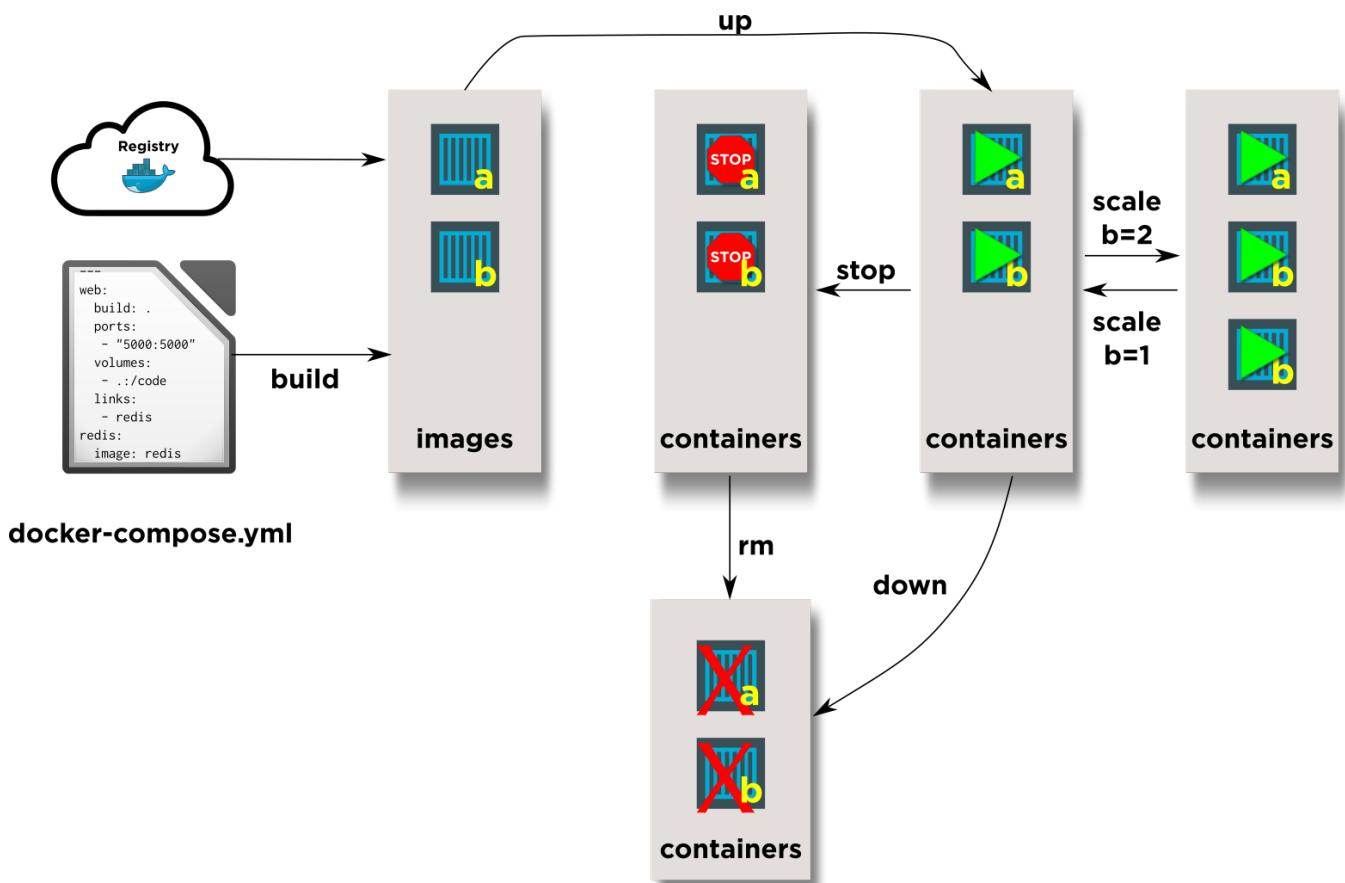
- `docker-compose scale <service>` scales a single service in the composition
- beware of statically published ports!



```
$ docker-compose scale web=2
WARNING: The "web" service specifies a port on the host. If multiple containers for
this service are created on a single host, the port will clash.
Creating and starting 2 ... error

ERROR: for 2  driver failed programming external connectivity on endpoint bas_web_2
(499e6082015a9448eb3d5fd65d3142cadacce8b0a78a32ba038c6dcaaac96b43): Bind for
0.0.0.0:5000 failed: port is already allocated
Removing bas_web_2 ... done
```

Composition lifecycle



Docker-compose v2 & v3

- requires Compose 1.6.0+ and Docker Engine 1.10.0+
- provides more flexibility for build, volumes, & network configuration
- creates a new network for each composition
- no need to link container since v2

```
---
version: '2'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    networks:
      - front-tier
      - back-tier
  redis:
    image: redis
    volumes:
      - redis-data:/var/lib/redis
    networks:
      - back-tier
volumes:
  redis-data:
    driver: local
networks:
  front-tier:
    driver: bridge
  back-tier:
    driver: bridge
```

Mixing build and image

On each service, you can set both the `image` and the `build` parameters. This allows to document/commit the name of the container image.

```
---
version: '2'
services:
  web:
    build: ./  
    image: camptocamp/web:1.0
    ports:
      - "5000:5000"
    networks:
      - front-tier
      - back-tier
networks:
  front-tier:
    driver: bridge
  back-tier:
    driver: bridge
```

```
$ docker-compose build --pull --no-cache  
$ docker-compose up -d  
$ docker-compose push
```

Restart policy

- Specify behaviour when service crash: restart service
- Restart conditions:
 - `on-failure`: When exit code $\neq 0$
 - `any`: always restart

```
version: "3.7"
services:
  redis:
    image: redis:alpine
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
```

HealthCheck

- Added in Compose v2.1
- Run command in container for healthcheck: `curl localhost:8080/healthcheck`
- Can be configured:
 - `interval`: time between healthcheck
 - `timeout`: how long to wait for response
 - `retries`: how many failure before restart
 - `start_period`: wait before first healthcheck

Lab 11.1: Use Docker-Compose



Objective

- Use Docker-Compose to run multiple containers

Steps

- Create a `docker-compose.yml`
- Add a service for backend
- Add a service for frontend
- Start the composition and watch the logs
- Add restart policy, remove `data.json` and check if container is restarted
- Optional: if you have deployed Portainer, check running containers and stacks

Questions

- How can you check that your composition is well deployed?
 - docker ps
 - docker-compose ps
 - ps axuw

Labs solutions

You can found `docker-compose.yml` for services [here](#).

Checkpoint: Docker-Compose

Composing containers

- Containers in compositions have a namespace prefix in their name:
 - Yes
 - No
- The Docker-Compose v2 format allows to
 - monitor services automatically
 - tune volumes
 - tune ports
 - tune networks
 - justify a migration project to your boss
- Scaling services
 - duplicates running containers
 - creates new containers from the base image
 - start new instances of the existing containers

DEVELOP WITH DOCKER-COMPOSE

Lesson 12: Development Workflow Using Docker-Compose

Objectives

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

Use `docker-compose` in the following scenarios:

- Build
- Debug
- Test
- Demo

Using Docker-Compose for Build

In order to build a container image you need:

- a context that contains the code
- the localisation of the `Dockerfile` that contains the build instructions
- the name of the image to produce

All this informations can be stored in the `build` section of the `docker-compose.yaml` file.

```
services:  
  backend:  
    build:  
      context: ./backend/python  
      dockerfile: Dockerfile  
      image: camptocamp/backend:latest  
    args:  
      PYTHON_VERSION: 3.9.5-slim
```



A URL to a Git repository can be used as context

Using Variables during Build

Build different versions using the same `Dockerfile`:

- Different versions of language SDK or libraries
- Build flags, theme, flavor
- Main application version if code is not in the same repository

```
FROM python:3.9.5-slim
```

```
...
```

```
ARG PYTHON_VERSION=3.9.5-slim  
FROM python:${PYTHON_VERSION}
```

```
ARG BACKEND_VERSION=1.2.0  
RUN git clone --branch ${BACKEND_VERSION} https://github.com/camptocamp/containers-cour  
...
```

Dockerfile refactor

Variable Scope

Using variables in `FROM` instructions

In order to use a variable in a `FROM` instructions, it needs to be defined *before*.

```
ARG PYTHON_VERSION=3.9.5-slim
FROM python:${PYTHON_VERSION} AS test
...
FROM python:${PYTHON_VERSION} AS build
...
```

Using variables in other instructions

Context is different for other instructions: you only have access to `ARG` defined within the current stage.

```
FROM python AS test
ARG STEP=test
RUN echo ${STEP}

FROM python AS build
ARG STEP=build
RUN echo ${STEP}

FROM python AS quality
# STEP is not defined
RUN echo ${STEP}
```

Variable Scope

Using variables for base image and during build

First `ARG` is for base image, the next one is for package installation.

```
ARG BASE_TAG=10
FROM postgres:${BASE_TAG}

ARG BASE_TAG

RUN apt-get update && apt-get upgrade -y && \
    apt-get install -y postgresql-$BASE_TAG-postgis-3 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
```

But value is set only one time on command line:

```
$ docker build --build-arg BASE_TAG=12 .
```

Lab 12.1: Add Build argument



Objective

- Use one `Dockerfile` to build several images

Steps

- Add argument to the `Dockerfile` for base image tag
- Add services in `docker-compose.yaml` to build several images:
 - Python version: `3.9.5-buster`, `rc-slim`, `3.6-stretch`
 - Go version: `1.15-stretch`, `1.16.4-buster`
 - Java version: `9`, `10`
- Test new images
- Optional: add another argument for a dependency version:
 - Python Flask version: `1.1.4`
 - Go
 - Java:

Questions

- Can you run specific command based on argument value?
 - Yes
 - No

Use multiple compositions

Compositions can be used for multiple purpose:

- build
- demo
- test

By default, docker-compose uses:

- `docker-compose.yml`
- `docker-compose.override.yml`

The `-f, --file FILE` flag allows to override compose files:

```
$ docker-compose -f docker-compose-build-all.yml build --pull --no-cache
$ docker-compose -f docker-compose-build-all.yml push

$ docker-compose -f docker-compose.yml -f docker-compose-benchmark.yml up
$ COMPOSE_FILE=docker-compose.yml:docker-compose-benchmark.yml docker-compose up
```



The project name can be set with the `-p` flag, and defaults to the current directory name

Lab 12.2: Override default composition



Objective

- Adapt composition to debug on host

Steps

- Create a `docker-compose-debug.yml`; don't modify `docker-compose.yml`
- Change the frontend configuration to use `172.17.0.1:8080` as backend address
- Disable the service backend by setting `replicas=0`
- Run backend on host (not in container)
- Test by using the frontend in composition
- Optional: create a `docker-compose-build-all.yml` to build the backend with different versions of the base image

Questions

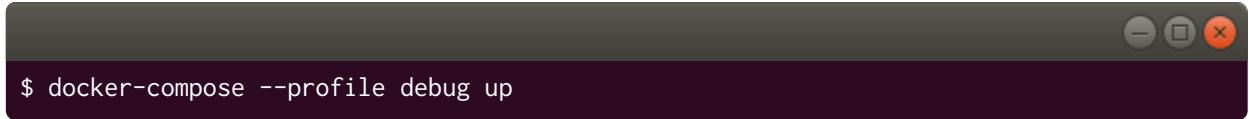
- Can you unset an existing environment variable in a service?
 - Yes
 - No
- Can you use a composition for automated build (CI/CD)?
 - Yes
 - No

Composition profiles

One or more *profiles* can be attached to a *service*:

- Services without profiles are always started
- Services with profiles are started only if the profile is specified

```
version: "3.8"
services:
  minio:
    image: minio/minio
    profiles:
      - demo
      - debug
  backend:
    image: camptocamp/backend
```



```
$ docker-compose --profile debug up
```



You can only activate services with profile, not deactivate always-on services.

Database Integration

Many applications use relational databases. How to bootstrap a database container with test data?

The PostgreSQL image has a nice **bootstrap procedure**:

- Check if there is already a database bootstraped on `/var/lib/postgresql`
- Initialize a new database using `POSTGRES_DB`, `POSTGRES_USER` and `POSTGRES_PASSWORD`
- Load content of `/docker-entrypoint-initdb.d/` based on extensions:
 - `.sh`: executed or sourced if not executable
 - `.sql` sent to the database with `psql` client
 - `.sql.gz`, `.sql.xz` decompressed and sent to the database



Lab 12.3: Add Database



Objective

- Add a test database

Steps

- Update `docker-compose-debug.yml` to deploy a new service for PostgreSQL
- Set up a volume on `/var/lib/postgresql`
- Set up database name, username and password
- Publish PostgreSQL port
- Put init scripts in `/docker-entrypoint-initdb.d/`:

```
CREATE TABLE product (id SERIAL, name TEXT, description TEXT, PRIMARY  
INSERT INTO product (name, description) VALUES ('geomapfish', 'GeoMap  
...  
...
```

Questions

- How can you simulate an S3 object storage bucket?
 - Use Postgis
 - Use MinIO
 - Use S3fs
 - Use Redis

Multistage: Select stage to run

When using multistage images, last stage is run by default.

The `build.target` field allows to run an intermediary stage:

```
services:  
  backend:  
    backend:  
      context: ./backend/go  
    debug:  
      context: ./backend/go  
    target: build
```

```
FROM golang AS build  
RUN go build  
  
FROM scratch  
COPY --from=build
```

Lab 12.4: Debug in Container



Objective

- Debug application in container
- Don't modify `Dockerfile` or `docker-compose.yml`

Steps

- Update `docker-compose-debug.yml` to use the first stage of the python backend
- Add a `volumes` section to the backend configuration to:
 - mount local code in the container
 - add missing `data.json`
- Override the entrypoint and command to:
 - install a debug component: `pip install debugpy`
 - launch app with debug: `python -m debugpy --listen 0.0.0.0:5678 api.py`
- Publish the debug port: 5678
- Connect VS Code to the debug socket

Questions

- What can be done for compiled languages?
 - Rebuild and relaunch
 - Use a special VS Code image to rebuild inside it

Labs solutions

You can find a implementation of labs [here](#).

Using Compose To Test Application

- Mock external services for tests as additional composition services:
 - Database
 - Object Storage
 - API
- Dedicated container to launch integration tests:
 - curl
 - JMeter
 - Gatling



You still need to write tests

Using Compose For Demo

Many open source projects include a `docker-compose.yml` to try the application.

- `docker-compose.yml` Start the application for demo or presentation.
It's the contributor entrypoint.
- `docker-compose-build.yml` Used by CI/CD
- `docker-compose-debug.yml` For development purposes

```
$ git clone https://github.com/camptocamp/terraboard.git  
$ cd terraboard  
$ docker-compose up -d
```


SERVICE DISCOVERY

Lesson 13: Service Discovery

Objectives

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

- Understand how service discovery enables dynamic configuration

Service auto-configuration

Dynamic scaling and scheduling requires dynamic configuration of:

- load balancers
- clusters
- monitoring
- backups
- etc.

Can be done with Configuration Management (e.g. Puppet's exported resources), but:

- not practical
- slow

How does it work?

Service discovery is based on key/value stores, usually distributed:

- Zookeeper
- Etcd
- Consul

Services announce themselves to the store. Consumer services retrieve announced services from the store.

Exercise 13.1: use haproxy with consul



Objective

- Use consul to automatically setup an haproxy load-balancer in front of services

Steps

- Clone <https://github.com/camptocamp/docker-consul-demo> and follow the instructions in the README

Questions

- What would be necessary to bring this setup to production?
 - An overlay network
 - Monitoring of the backends
 - A consul cluster with 3+ nodes
 - A distributed load-balancer

Service discovery in production

- Often hard to setup (cluster with quorums => at least 3 nodes per site)
- Orchestrators (such as Rancher) provide it

Checkpoint: Service Discovery

When orchestrating services with discovery:

- Service Discovery is useful for
 - Monitoring
 - Data backups
 - Proxy configuration

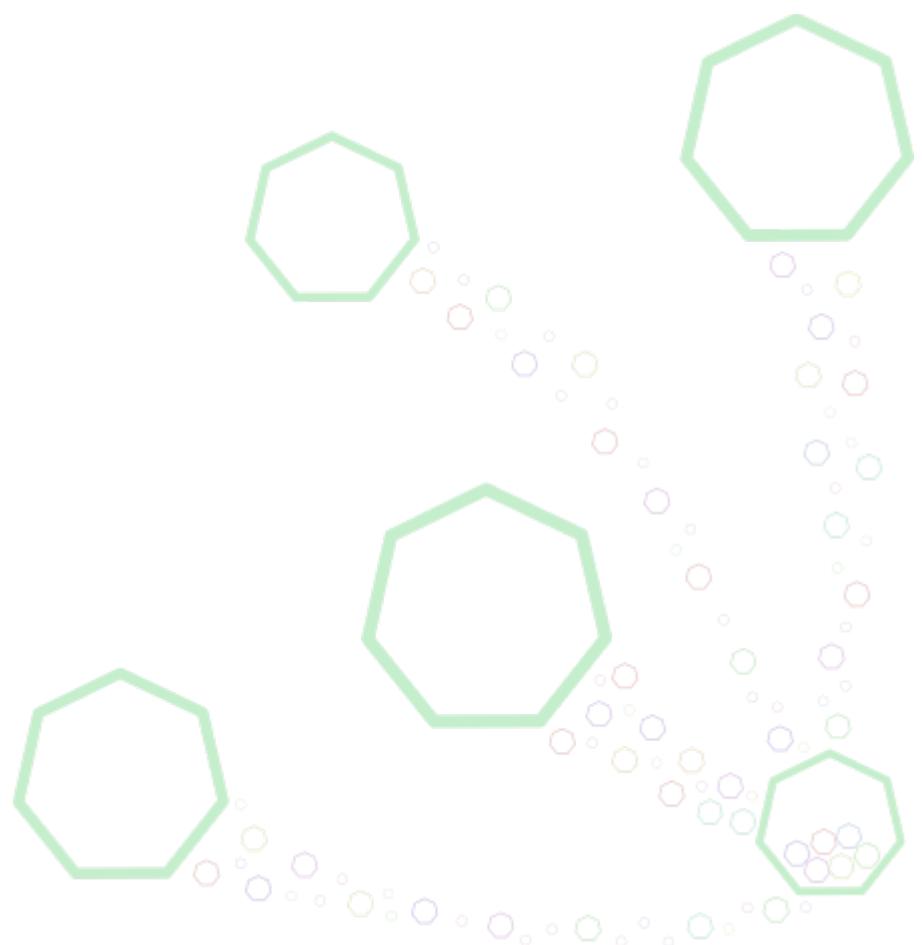
OBSERVABILITY

Lesson 14: Observability

Objectives

At the end of this lesson, you will know about:

- SLA, SLO & SLI
- What is observability
- Why we need new observability tools



Service Levels

Service Levels guarantee the Quality of Service.

Service Level Agreements (SLA)

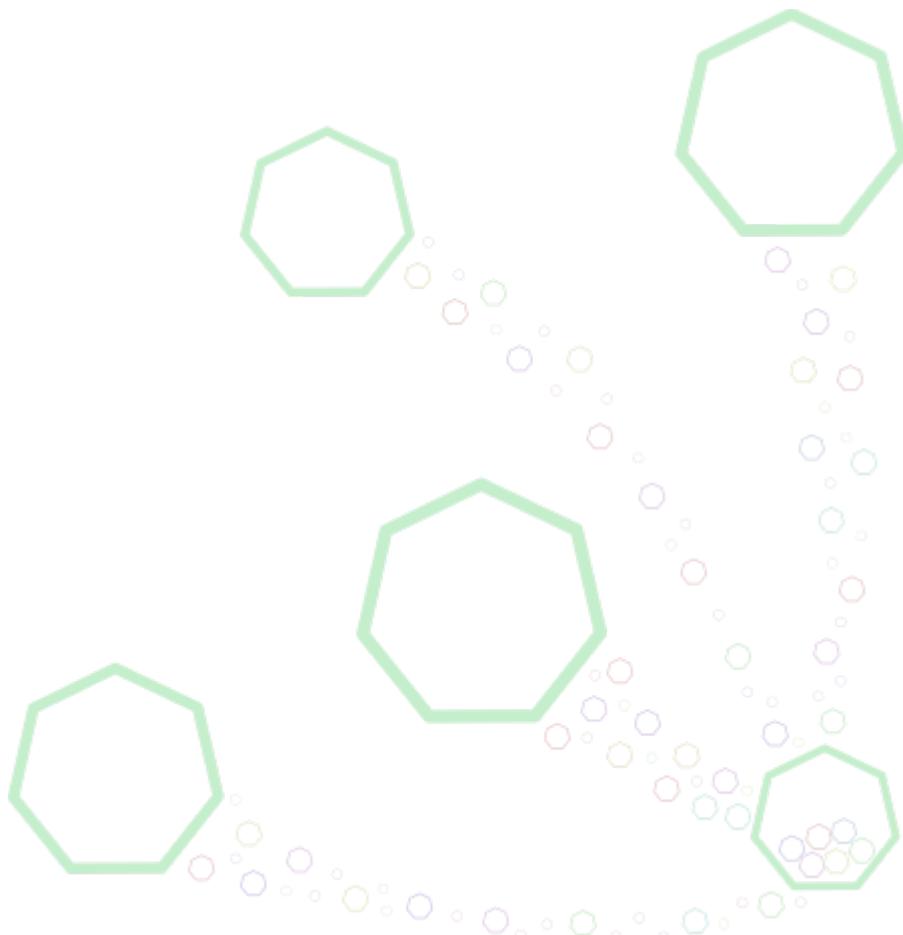
- contractual commitments
- not technical, focused on business
- financial compensation

Service Level Objectives (SLO)

- technical objectives to achieve SLA

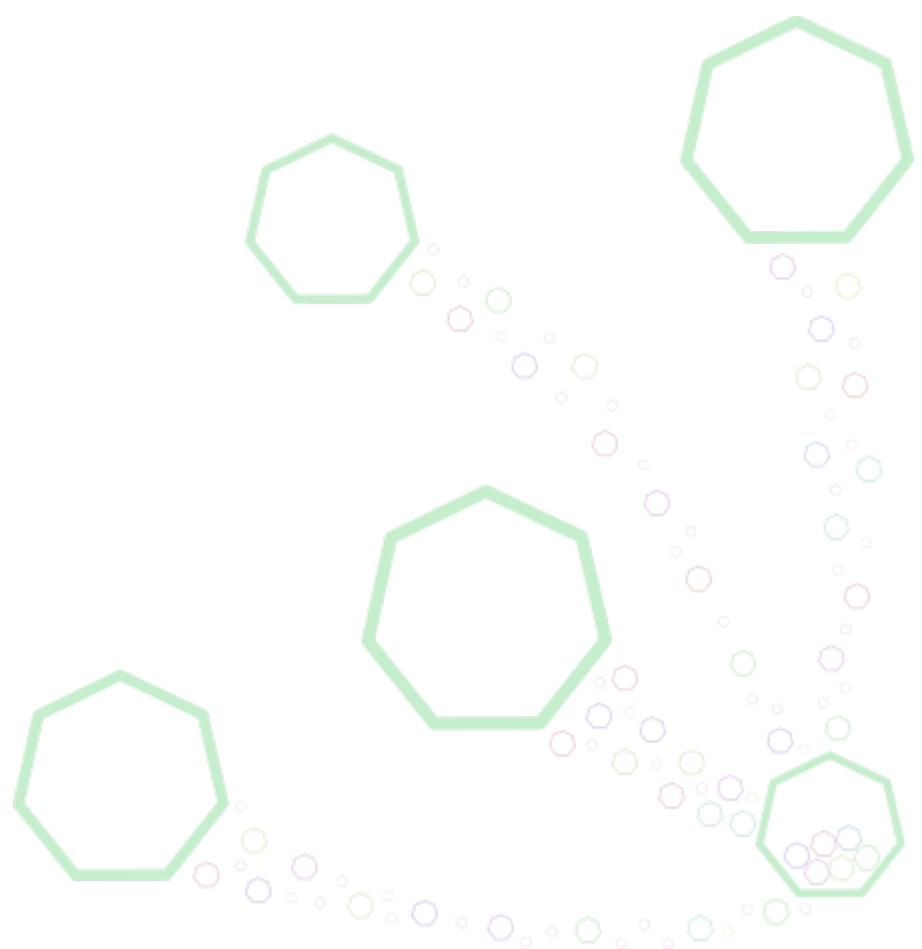
Service Level Indicators (SLI)

- indicators used to measure SLO adequation



Don't aim for 100%

- SLA never aim for 100% (max is the min infra SLA)
- SLO represent an error budget
- SLI let you measure the quality, so you can plan how to use the budget



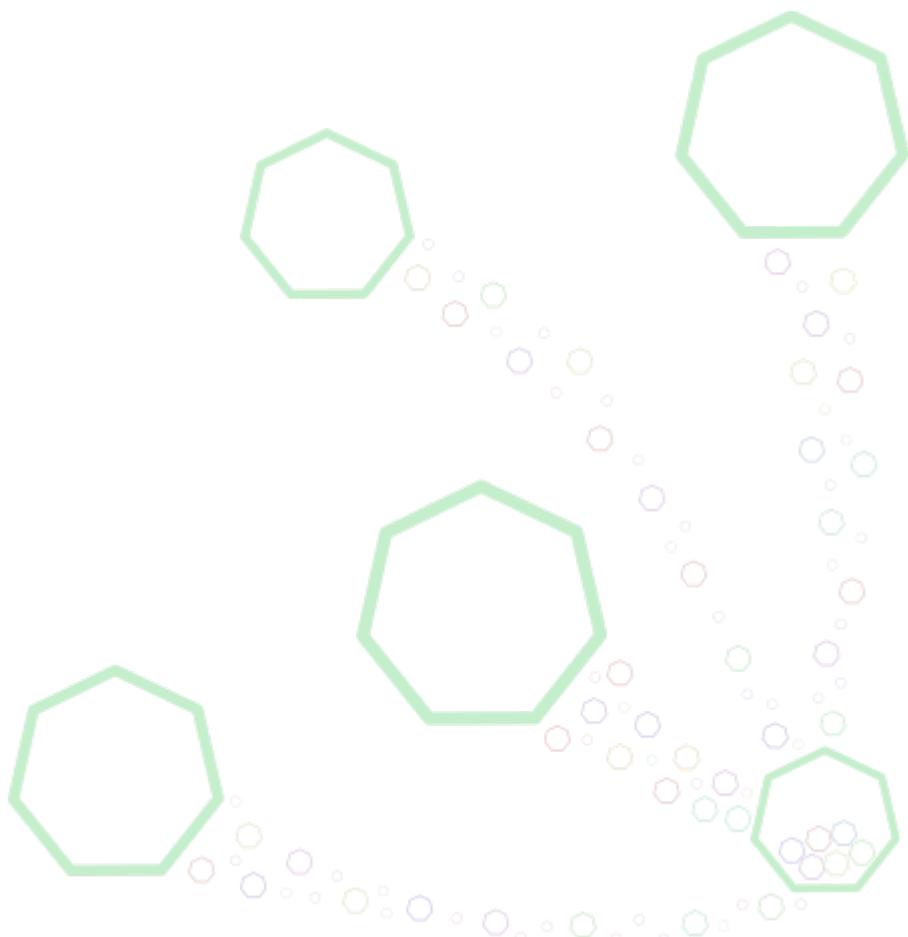
Observability



In control theory, observability is a measure of how well internal states of a system can be inferred by knowledge of its external outputs. - Wikipedia: Observability

Comprises:

- Metrics
- Graphing
- Monitoring
- Logs
- Alerting

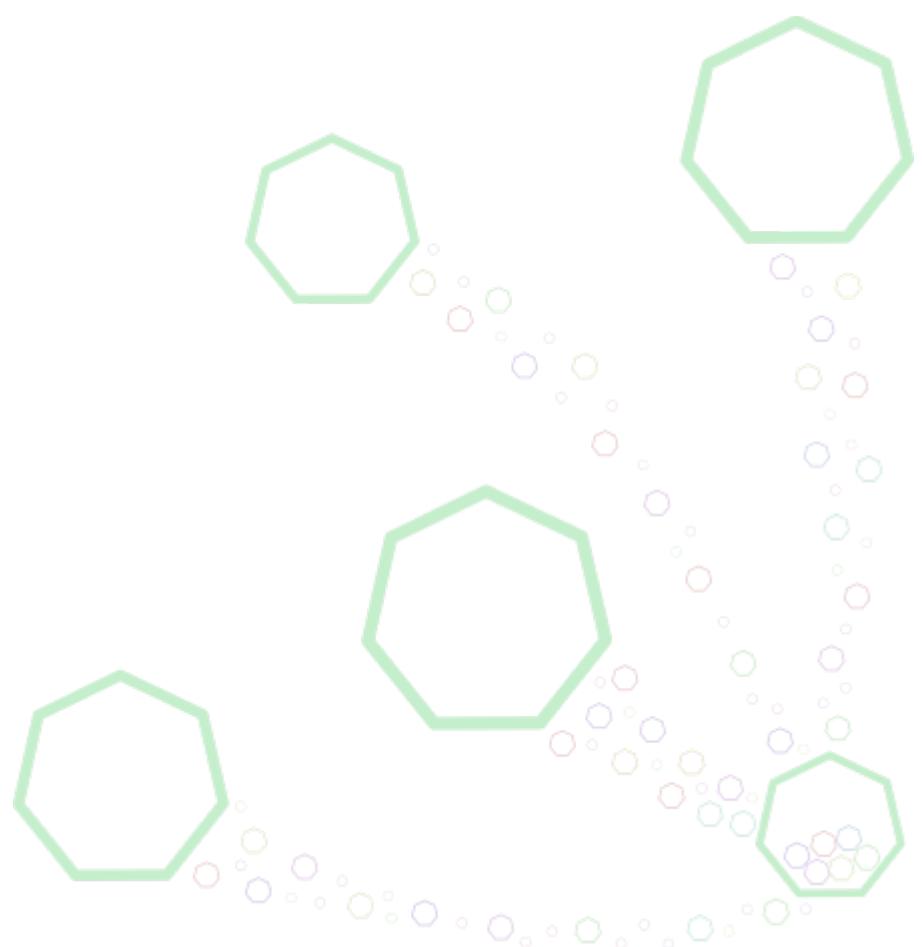


Dumb Monitoring

When responsibility is linked to power.

Operations monitor the wrong metrics:

- Disk space
- CPU load
- Free memory / Swap
- Network load

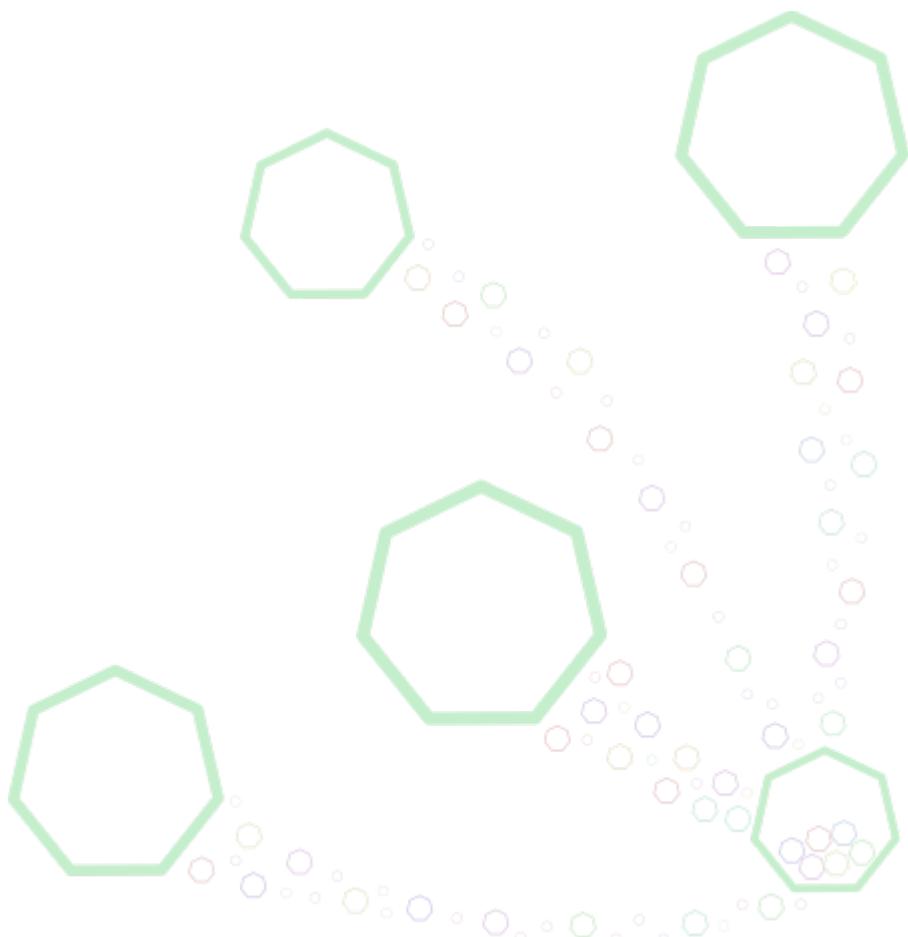


Smart Monitoring

When responsibility is linked to knowledge.

Developers can observe:

- Internal metrics (% errors, cache, replication state)
- Application logic (clients, sessions)

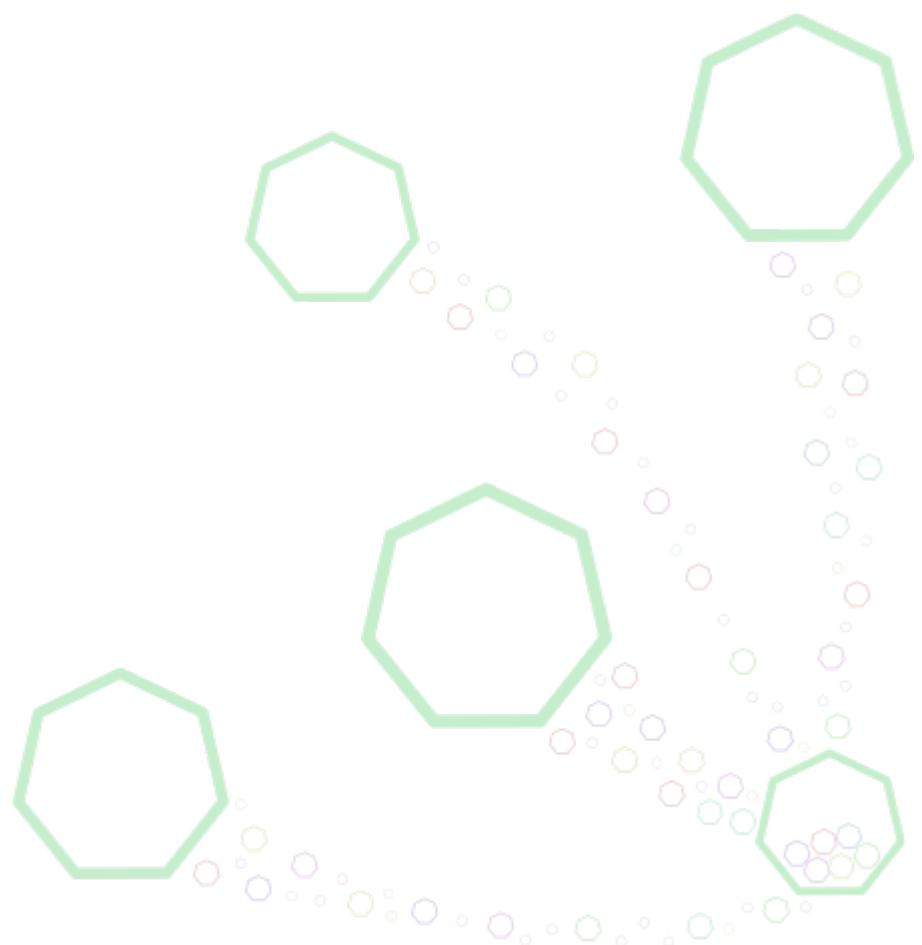


Nodes vs Apps

Applications on the cluster are not linked to Nodes:

- They can split into many microservices
- They can be spawned/respawned on any Node
- They can scale at will
- We want logic & global observability

We need aggregated logical metrics



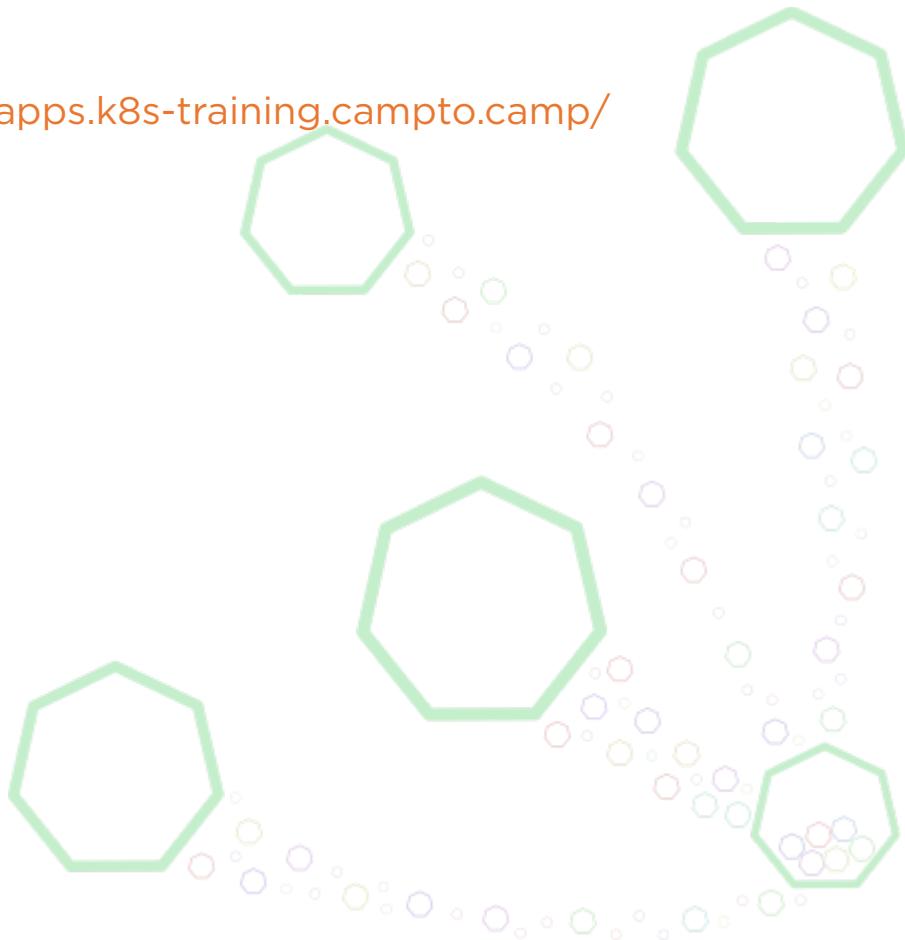
Prometheus

- Made for Container monitoring
- Inspired by Google Borgmon (tool to monitor Borg)
- Second CNCF project (after Kubernetes)



- Time series
- Labels
- Federation

Demo: <https://prometheus.apps.k8s-training.camto.camp/>

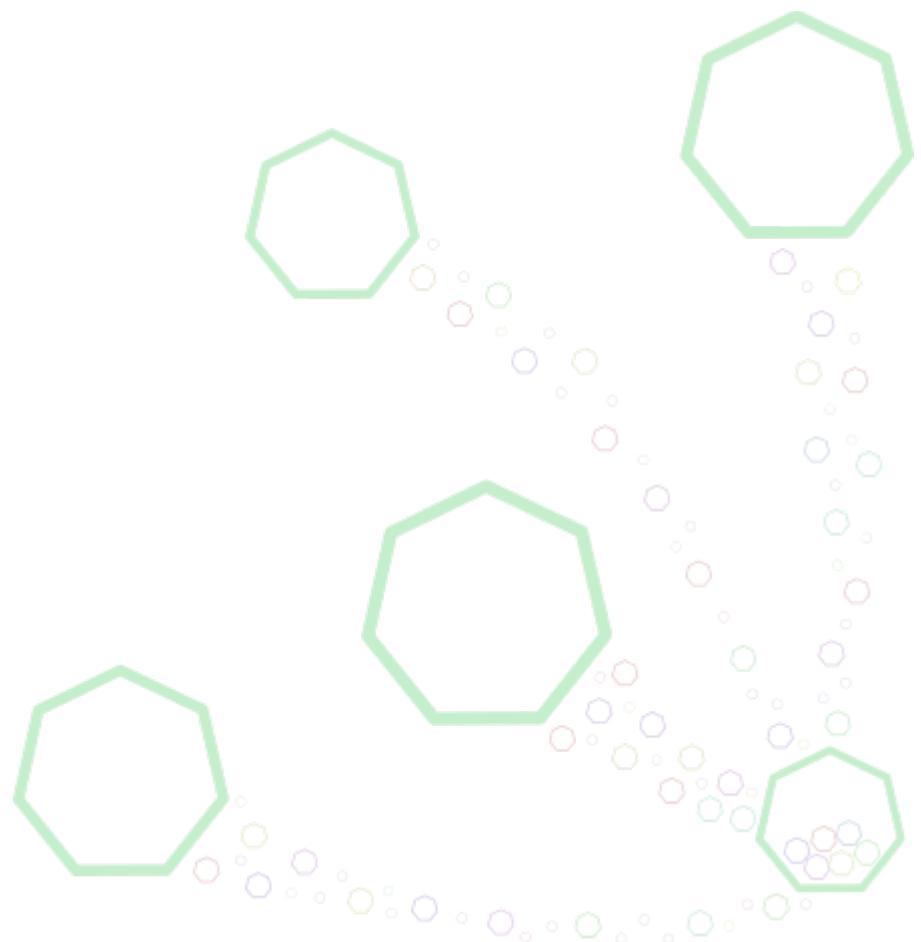


Grafana

- Web interface for graphs
- Supports multiple data sources (including Prometheus)



Demo: <https://grafana.apps.k8s-training.campto.camp/>



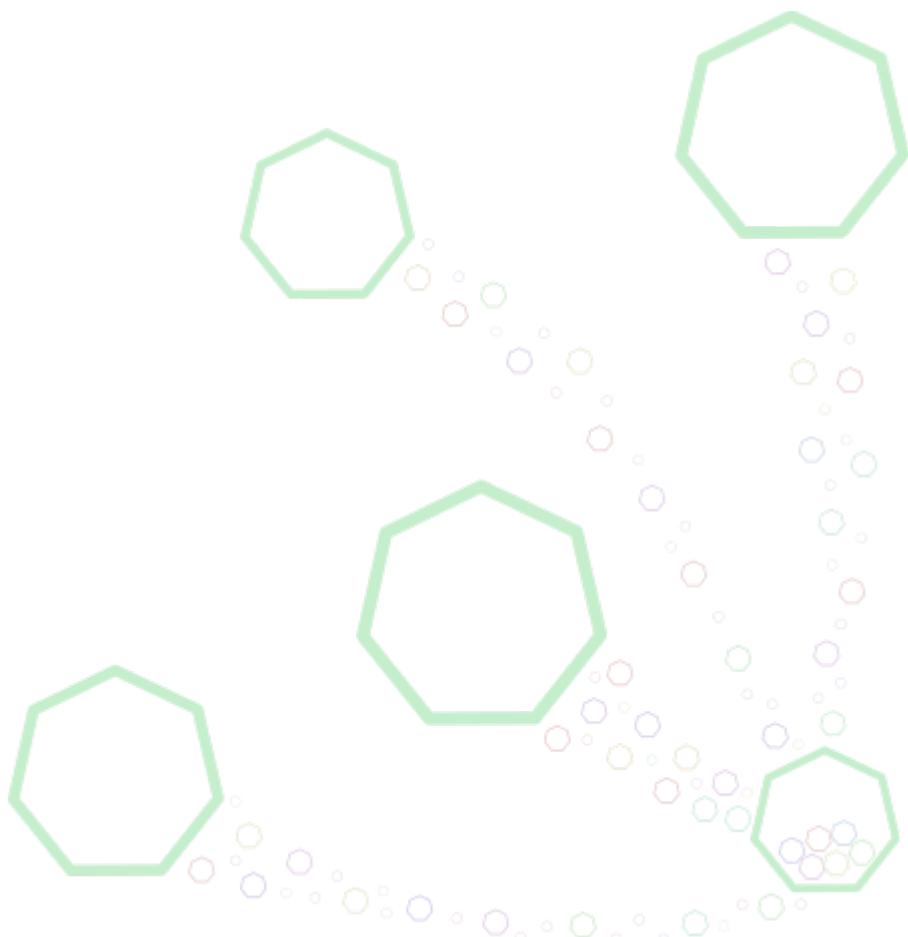
Alert Manager

Alerts defined in Prometheus (along with recording rules):

<https://prometheus.apps.k8s-training.campto.camp/rules>

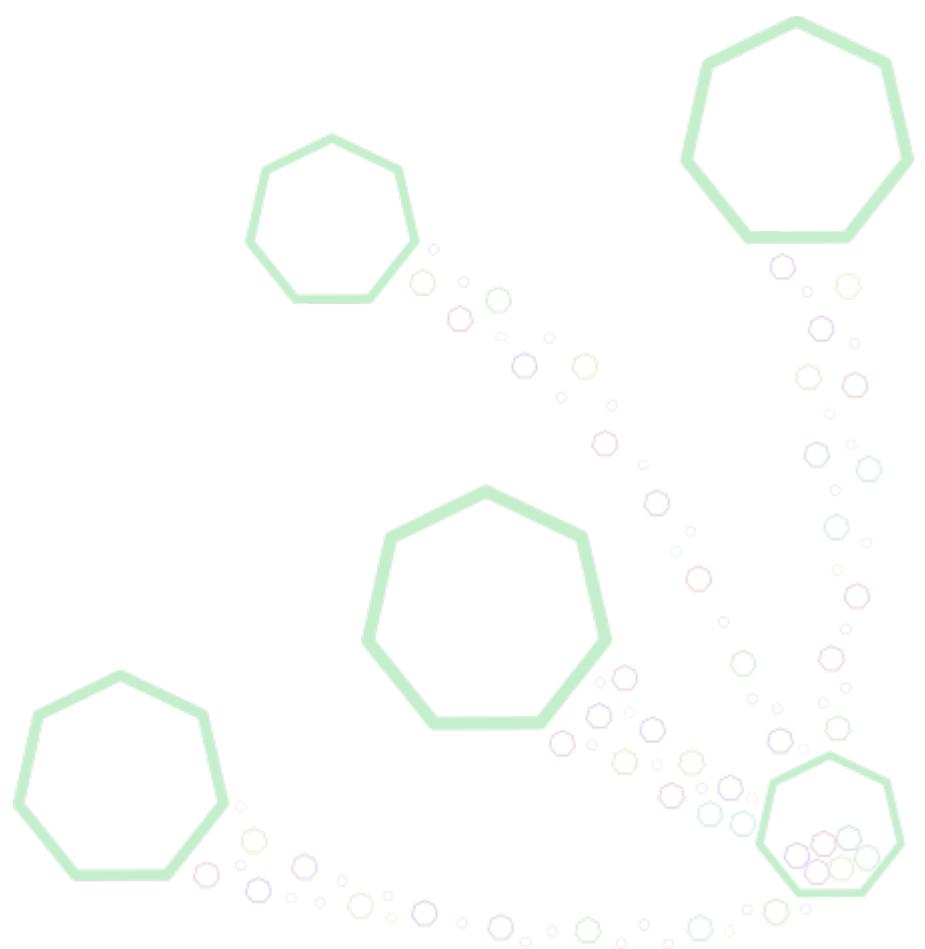
- Can be linked to Data source
- Can be silenced

Demo: <https://alertmanager.apps.k8s-training.campto.camp/>



Loki

- Logs
- Compatible with Grafana

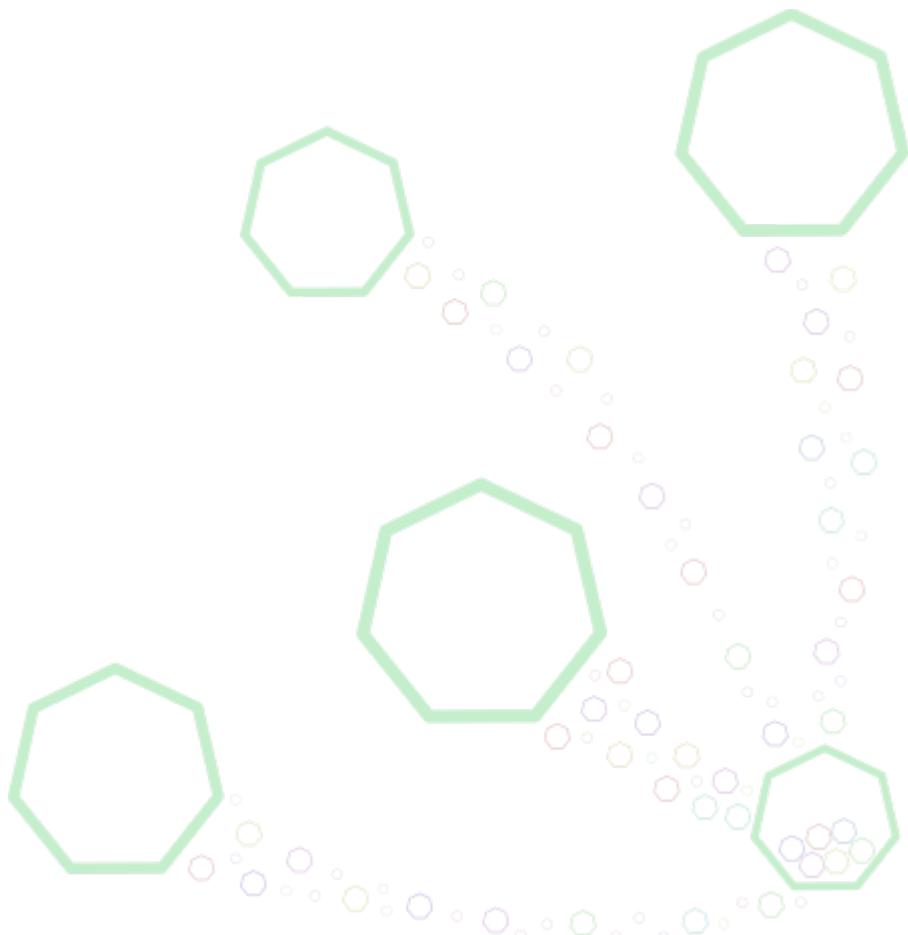


Demo: Observability



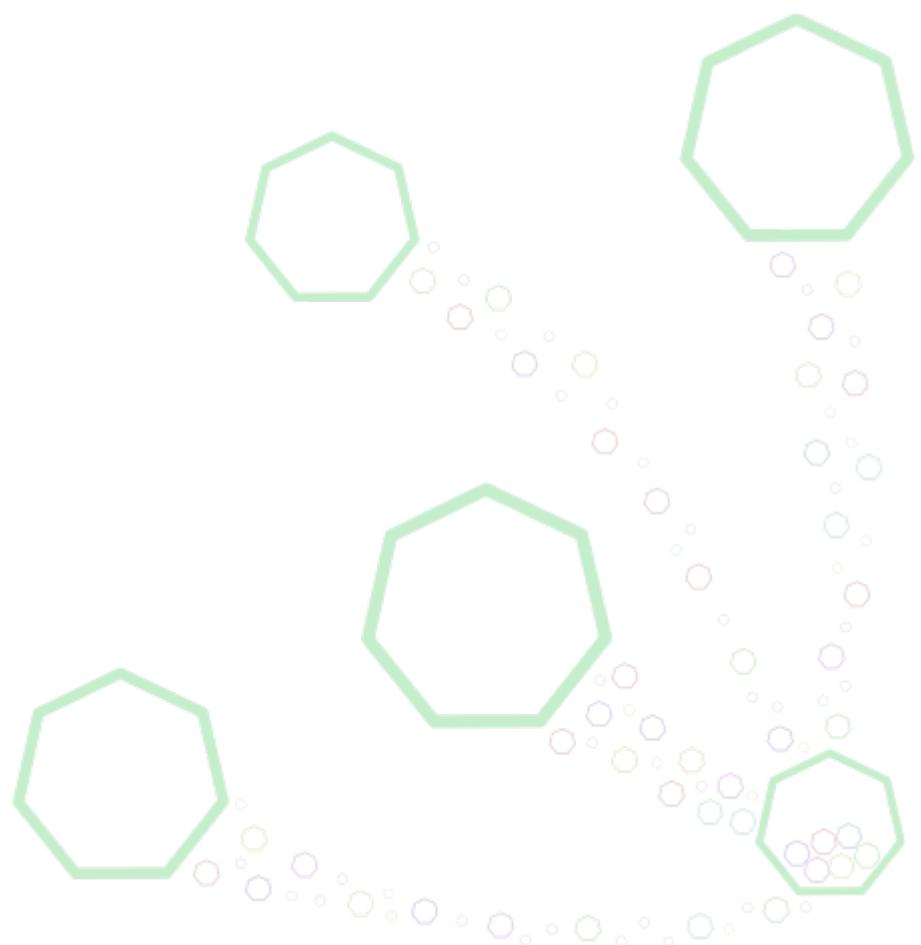
The instructor will demonstrate:

- Deploying the stack (Prometheus + Grafana) with docker-compose
- Integrating an app's metrics into the stack



Checkpoint: Observability

- Time-Series based observability can replace
 - On/Off Monitoring
 - Backups
 - RRD-based graphs
 - SMS Paging Alerts
- Metrics can come from
 - Node Operating Systems
 - Middleware Components
 - Network Components
 - Applications
 - Logs
- How to monitor text values
 - Turn them into discrete enumerables
 - Use Hash function or monitor its length
 - Use logs instead of metrics
 - Use Nagios



APPLICATION MONITORING

Lesson 15: Application Monitoring

Objectives

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

- Implement metrics in applications
- Build dashboards to visualize metrics

Prometheus Exporter Integration

It is better to integrate the exporter code in the application.

Sometimes it's not possible:

- you cannot modify the application, you're not maintainer
- the application already exposes some metrics, but not in the Prometheus format.

You can create a separate application that will act as an Adapter to monitor the main application.

Prometheus Text Based Format

The exporter needs to generate metrics in the [Prometheus Format](#):

- metrics name
- metrics type and description
- one or more labels
- value

```
# HELP http_requests_total The total number of HTTP requests.  
# TYPE http_requests_total counter  
http_requests_total{method="get",code="200"} 1234027  
http_requests_total{method="post",code="200"} 1027  
http_requests_total{method="post",code="400"} 3
```

Metrics Type and Name

Metric Type

There are several **types of metrics**, most metrics use `counter` or `gauge`:

- `counter` metrics are always incremental, cumulative, e.g.: error count, total requests
- `gauge` metrics can increase and decrease, e.g.: connected customers, disk usage

Metric Name

The name of the metrics should:

- start with the exporter name (single word)
- use base units: seconds not milliseconds
- have a suffix describing the unit (plural form: `seconds`)

For example: `node_memory_usage_bytes`

- `node` is the name of the exporter and the component monitored
- `bytes` is the unit

Metrics Labels

Create multiple metrics with the same name which refer to different elements.

- Labels add dimensions to metrics.
- Labels should have a bounded sets of values

Some examples:

```
# HELP disk_usage_percent Usage of disk in percent (0-100)
# TYPE disk_usage_percent gauge
node_disk_usage_percent 63.7
```

Use labels for partitions:

```
# HELP disk_usage_percent Usage of disk in percent (0-100)
# TYPE disk_usage_percent gauge
node_disk_usage_percent{partition="/" 63.7
node_disk_usage_percent{partition="/var"} 37.2
node_disk_usage_percent{partition="/tmp"} 12.5
```

Multiple Metrics Labels

For HTTP request metrics, you can use labels for:

- the HTTP method: `method="POST"`
- the path: `path="/assets/script.js"` (maybe too many values, see also Tracing)
- the browser type: `browser="Firefox"`

For a store, you can build a sales-related metric and use labels for:

- the item category: `category="Tools"`, `category="Garden and Outdoor"`
- the brand: `brand="Makita"`, `brand="Weber"`

You can also create a metric for the stock with the same labels.

Prometheus can correlate metrics based on labels.

Metrics Labels – Full Example

```
# HELP disk_usage_percent Usage of disk in percent (0-100)
# TYPE disk_usage_percent gauge
node_disk_usage_percent{partition="/" 63.4
node_disk_usage_percent{partition="/var"} 37.6
node_disk_usage_percent{partition="/tmp"} 12.3

# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{method="GET", code="200"} 1234027
http_requests_total{method="POST", code="200"} 1027
http_requests_total{method="POST", code="400"} 3

# HELP store_sales_total The total number of sales.
# TYPE store_sales_total counter
store_sales_total{category="Tools", brand="Makita"} 163376
store_sales_total{category="Tools", brand="Bosh"} 298425
store_sales_total{category="Garden and Outdoor", brand="Weber"} 18346
store_sales_total{category="Garden and Outdoor", brand="Hyundai"} 163376

# HELP store_stock The number of stock items.
# TYPE store_stock gauge
store_stock{category="Tools", brand="Makita"} 234
store_stock{category="Tools", brand="Bosh"} 456
store_stock{category="Garden and Outdoor", brand="Weber"} 27
store_stock{category="Garden and Outdoor", brand="Hyundai"} 45
```

Example request:

```
store_stock - rate(store_sales_total[24h]) * 3600 * 24
```

Lab 15.1: Use Monitoring Stack



Objective

- Implement new metrics

Steps

- Checkout the `observability_demo` Git branch of the demo app
- Start the composition
- Connect to Prometheus and Grafana
- Generate some traffic with `watch -n 1 "curl localhost:8080/..."`
- Implement a new metric for unknown products (product not in `data.json`)
- Update the dashboard and commit the dashboard definition

Questions

- Which labels would make sense for the new metric?
 - Product Name
 - HTTP Referer
 - Action (buy/view)
 - Source IP address

Prometheus Libraries

A prometheus library exists for **most of programming languages**.

These libraries help in metrics implementation:

- Counter object with `inc()` method, Gauge with `set()`, `inc()` and `dec()` methods
- Create http endpoints for metrics, handle labels

For Python, the Prometheus library provides decorators for:

- monitoring raised exceptions
- parallel calls to functions
- time spent in functions

```
@counter_exception.count_exceptions()
def f():
    pass

@gauge_calls.track_inprogress()
def f():
    pass

@s.time()
def f():
    pass
```

Maintain metrics values up-to-date

The first approach is to update metrics values for each event that needs to be monitored.

- Keep metrics values up-to-date everytime
- Each time something changes and needs to be monitored, update the corresponding metrics
- Use `inc()`, `dec()`, `set()`

Benefits:

- Avoid complex and time consuming process to compute metrics values
- Changing scrape period should not impact application performance

Disavantages:

- Code to maintain metrics is running even if no metrics are fetched
- Code to maintain metrics is not centralized

```
view_metric = Counter('view', 'Product view', ['product'])

@app.route('/view/<id>')
def view_product(id):
    # Update metric
    view_metric.labels(product=id).inc()
    return "View %s" % id
```

On-Demand metrics

In this second approach, metrics are only computed when Prometheus asks for them.

- Metrics are computed on-the-fly during Prometheus request
- Use callbacks linked to metrics object

Benefits:

- No code is run if there is no metrics requests
- Centralized code

Disadvantages:

- Complexity of implementation
- Scrape period can impact the application performance

```
stock_metric = Gauge('stock', 'Stock count')
stock_metric.set_function(compute_stock)

def compute_stock():
    res = query('SELECT count(*) FROM product_stock')
    for line in res:
        return line[0]
```

HTTP Endpoint

Prometheus needs an HTTP endpoint to retrieve metrics. Prometheus libraries provide a light HTTP server to expose metrics:

```
from prometheus_client import start_http_server, Counter
view_metric = Counter('view', 'Product view', ['product'])
view_metric.inc()
start_http_server(8000, addr='0.0.0.0')
```

You can integrate metrics in a web application:

```
from prometheus_client import generate_latest
@app.route('/metrics')
def metrics():
    return generate_latest()
```

Lab 15.2: Use Prometheus Library



Objective

- Re-implement metrics using a Prometheus Library

Steps

- Load library
- Create counters
- Update counters
- Integrate with existing HTTP server
- Optional: implement a metric to count products using a callback (read items count in `data.json`)

Questions

- Can we add a label with product name on the new metric?
 - Yes
 - No

PRODUCTION CHALLENGES

Lesson 16: Production Challenges

Objectives

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

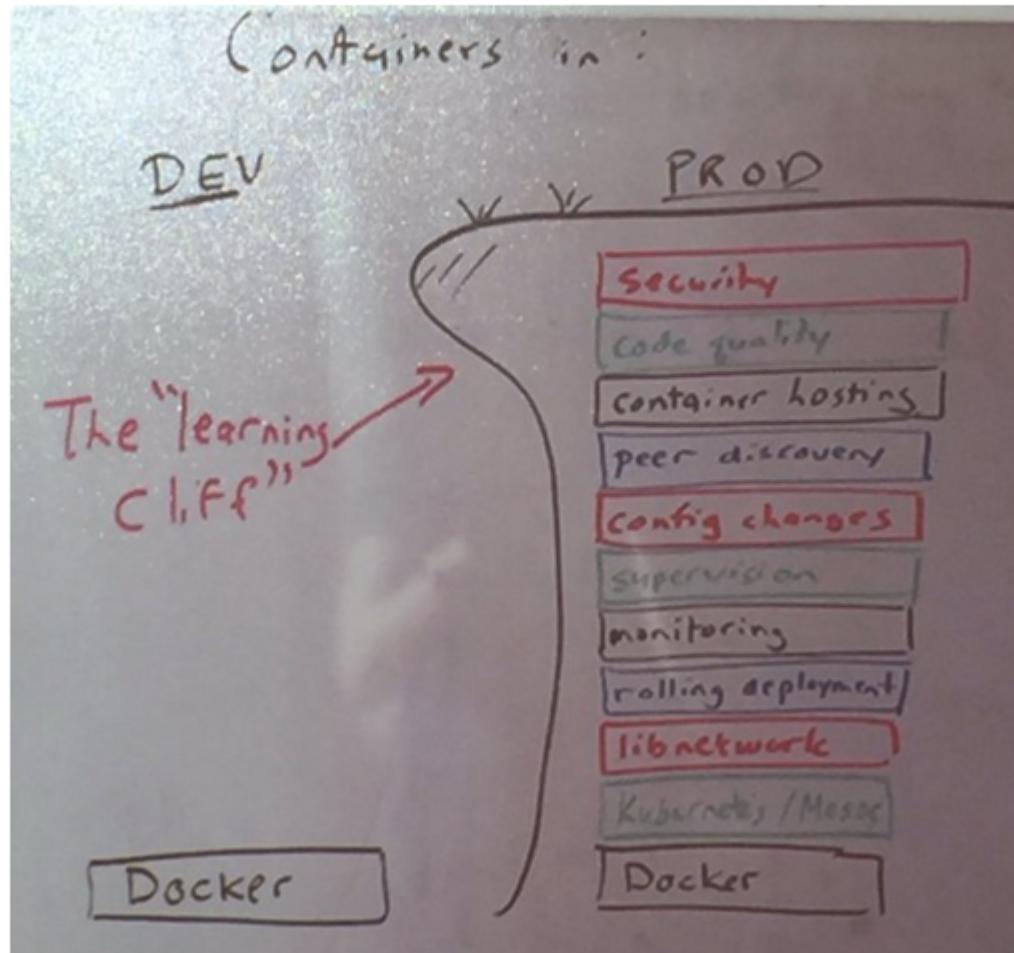
- understand the challenges of using Docker as a production environment
- know the available solutions for Docker orchestration

Dev/Prod gap

 Raphaël Pinson Retweeted



Michael Ducey @mfdii · Feb 10
Containers in Dev vs Prod



  583  433 

 Raphaël Pinson Retweeted



Kris Buytaert @KrisBuytaert · Feb 10
. @mfdii that's a very nice picture of the traditional dev-ops gap ...

  6  6 

[View conversation](#)

Config Management vs Immutable Infrastructure

- Long cycle management vs "trashable"
- Idempotent vs immutable
- Rebuild from scratch with every change
- Data must be kept aside

Multi-host (clustered) setup

- communication between containers on different (dynamic) hosts
 - overlay networks
- dynamic IPs
 - dynamic DNS updates is a requirement
- data location and persistence
- shared data between containers on different hosts

Data persistence

- named volumes solve the problem on one node
- multiple hosts require either:
 - tagging the host for affinity
 - or synchronizing data
 - or distributed/network filesystems (NFS/glusterfs/etc.)
 - or using network volume driver (convoy/flocker)

Service Discovery

- containers have dynamic IPs
- load-balancers need autoconfiguration
- requires a key-value system for service discovery (zookeeper/etcdb/consul/etc.)
- requires a key-value aware load-balancer (using consul-template/confd/etc.)

Security

- libraries are statically shipped in images
 - need to rebuild images frequently/automatically
- entrypoint launched as root by default
 - switch user in Dockerfile (using the `USER` directive)
 - or switch user in the entrypoint script
- storing secrets and distributing them
 - environment variables
 - files mounted as volumes
 - labels
 - key/value systems (vault)

Industry standard solutions

- Kubernetes
- Mesos/Mesosphere
- AWS ECS
- Docker Swarm
- Rancher
- Openshift

Kubernetes

- made by Google (project Borg)
- not specifically made for Docker
- heavy and powerful
- several distributions (e.g. Rancher, OpenShift)

Mesos/Mesosphere

- distributed compute cluster
- supports more than just Docker containers
- presents the cluster as a distributed Operating System

AWS ECS

- Amazon Service for Docker containers
- limited features

Docker Swarm

- official clustering solution
- native support for Docker API and Compose
- may not outlive Kubernetes

Rancher

- open-source project
- Docker platform
- simple approach
- support for Compose (rancher-compose)
- v1
 - comes with own orchestrator: Cattle
 - supports Swarm, Kubernetes and Mesos as alternate orchestration frameworks
- v2 supports Kubernetes only

Openshift

- RedHat solution for container platform
- Provides PaaS or dedicated instance
- Several versions:
 - local single host version: Minishift
 - Open-Source version: OKD
 - full version with Red Hat support: OpenShift container platform

Checkpoint: Production Challenges

When running Docker in production:

- What are problems linked to multi-nodes setups?
 - Horizontal scaling
 - Sharing data volumes
 - Monitoring
 - Ports
 - Managing secrets
 - Networks
 - DNS names pointing to the right IPs

KUBERNETES BASICS

Lesson 17: Kubernetes Basics

Objectives

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

- understand the need for container orchestration
- know about the history of Kubernetes



What is Kubernetes?



Kubernetes (commonly stylized as K8s) is an open-source container-orchestration system for automating deployment, scaling and management of containerized applications. ([Wikipedia > Kubernetes](#))

- Container Orchestrator
- Written in Go
- Uses YAML/JSON for configuration
- Supports Docker as Container Engine (among others)



Why an Orchestrator?

- Multiple Hosts
- Scaling
- Resilience
- Deployment Strategy (rollout, blue/green)
- Service Discovery
- Deployment Platform/Abstraction Layer: Pet vs Cattle
- Storage Management
- Self Healing

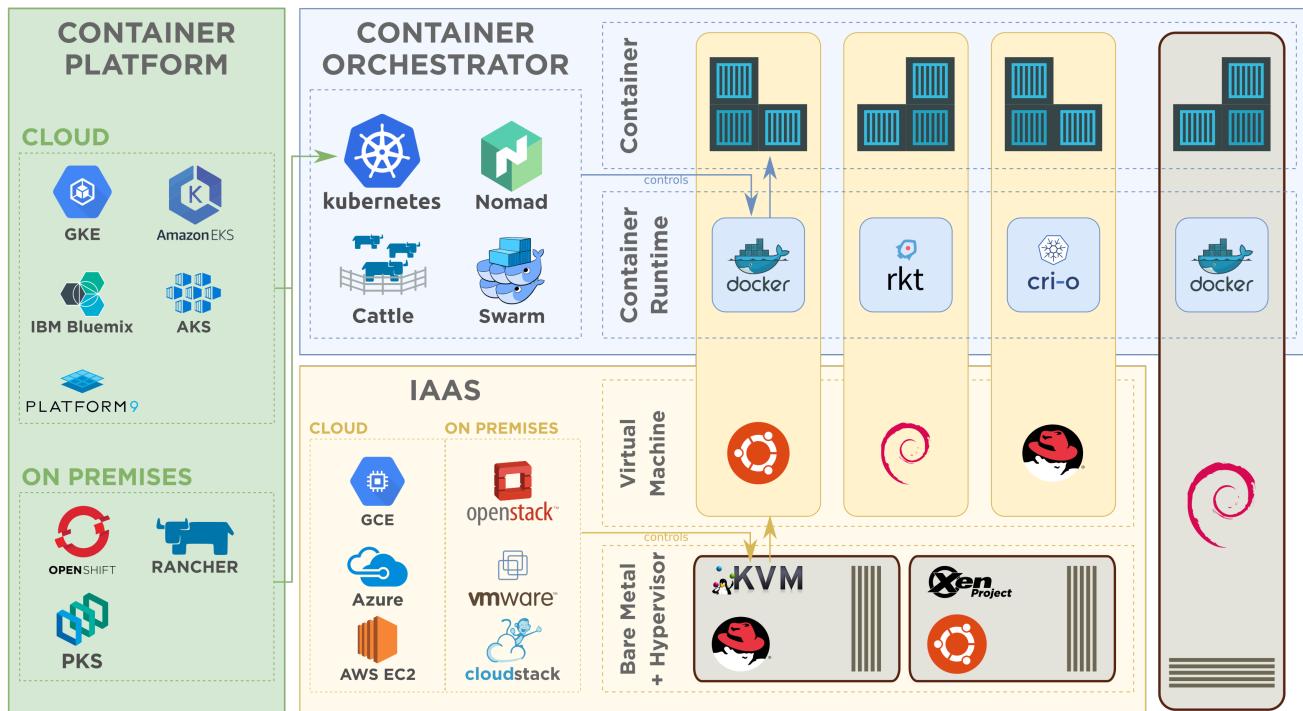


Container Orchestrators

- Kubernetes
- Swarm
- Mesos
- Nomad
- Cattle (legacy)
- ...



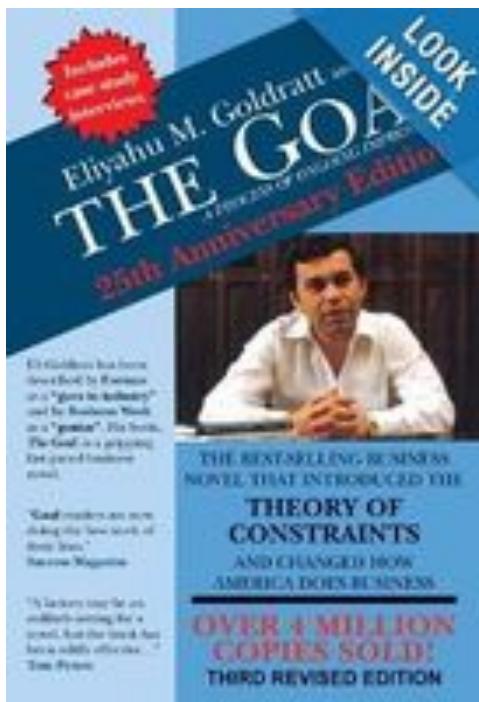
Container Orchestration Overview



DevOps

- Bridging Operations and Developers
- Three Ways of DevOps:
 - Systems Thinking
 - Amplify Feedback Loops
 - Culture of Continual Experimentation And Learning

More on **IT Revolution**. Recommended reading:



From the authors of *The Visible Ops Handbook*



The Phoenix Project

A Novel About IT, DevOps,
and Helping Your Business Win

Gene Kim, Kevin Behr, and George Spafford

Immutable Infrastructure



Immutability A pattern or strategy for managing services in which infrastructure is divided into “data” and “everything else”. “Everything else” components are replaced at every deployment, with changes made only by modifying a versioned definition, rather than being updated in-place. ([HighOps](#))



Kubernetes and immutability

Kubernetes is made to work in a transient way:

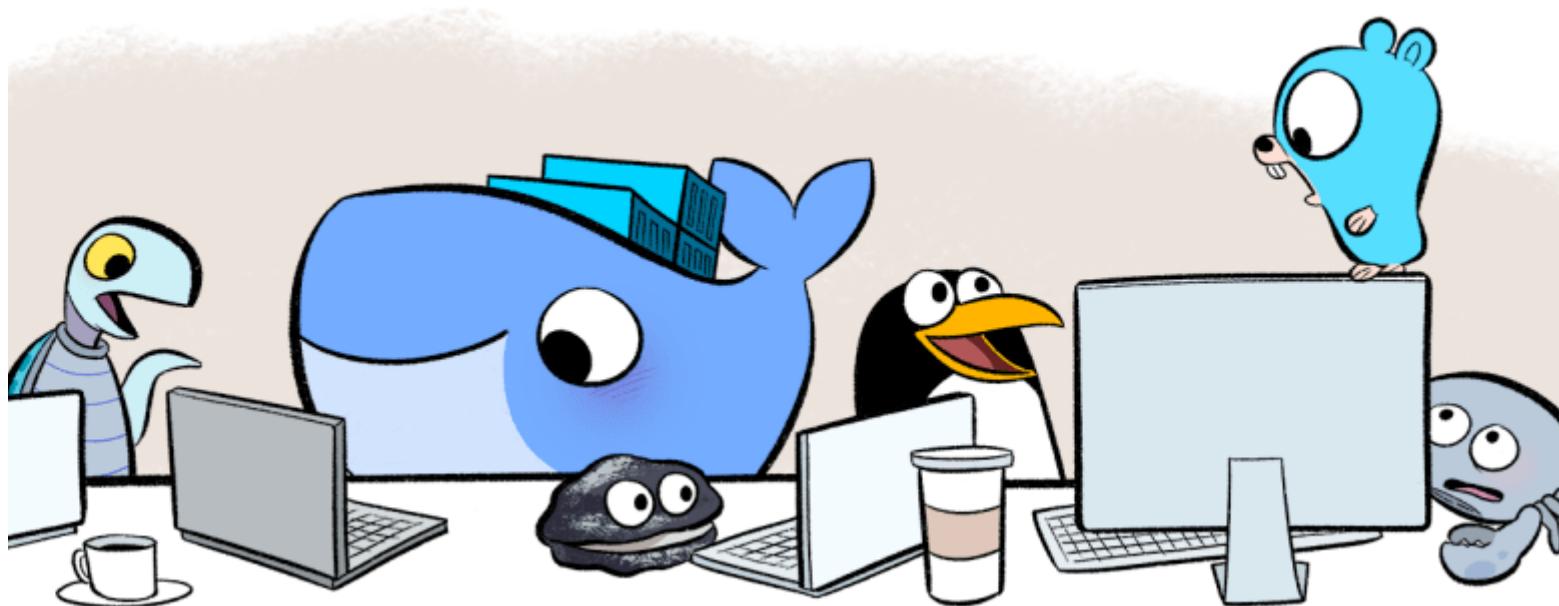
- objects are meant to be terminated and replaced by new versions
- this implies immutable operations
- containers as microservices are strongly encouraged



Supported Container Engines

- Docker
- Rkt
- RunC / containerd
- Cri-o
- etc.

Docker is the easiest to set up.



Kubernetes Solutions

Learning

- Minikube
- Minishift
- MicroK8s
- K3s
- Kind

Production

- Managed: GKE, EKS, AKS, PKS, OpenShift Online, etc.
- Cloud/On Premise: OpenShift, RKE, PKS, etc.

OpenShift

A Kubernetes-based Platform-as-a-Service

- By RedHat
- Managed or On-premise
- Based on K8s since v3
- Adds new objects (for Builds, Images, Routes)
- Better right management (RBAC)
- Better security (thanks to default SELinux configuration)
- CI/CD integration (Jenkins/GitLab)



Container Orchestrators at Google



Borg

- First unified container-management system at Google
- Made use (and contributed to) cgroups

Omega

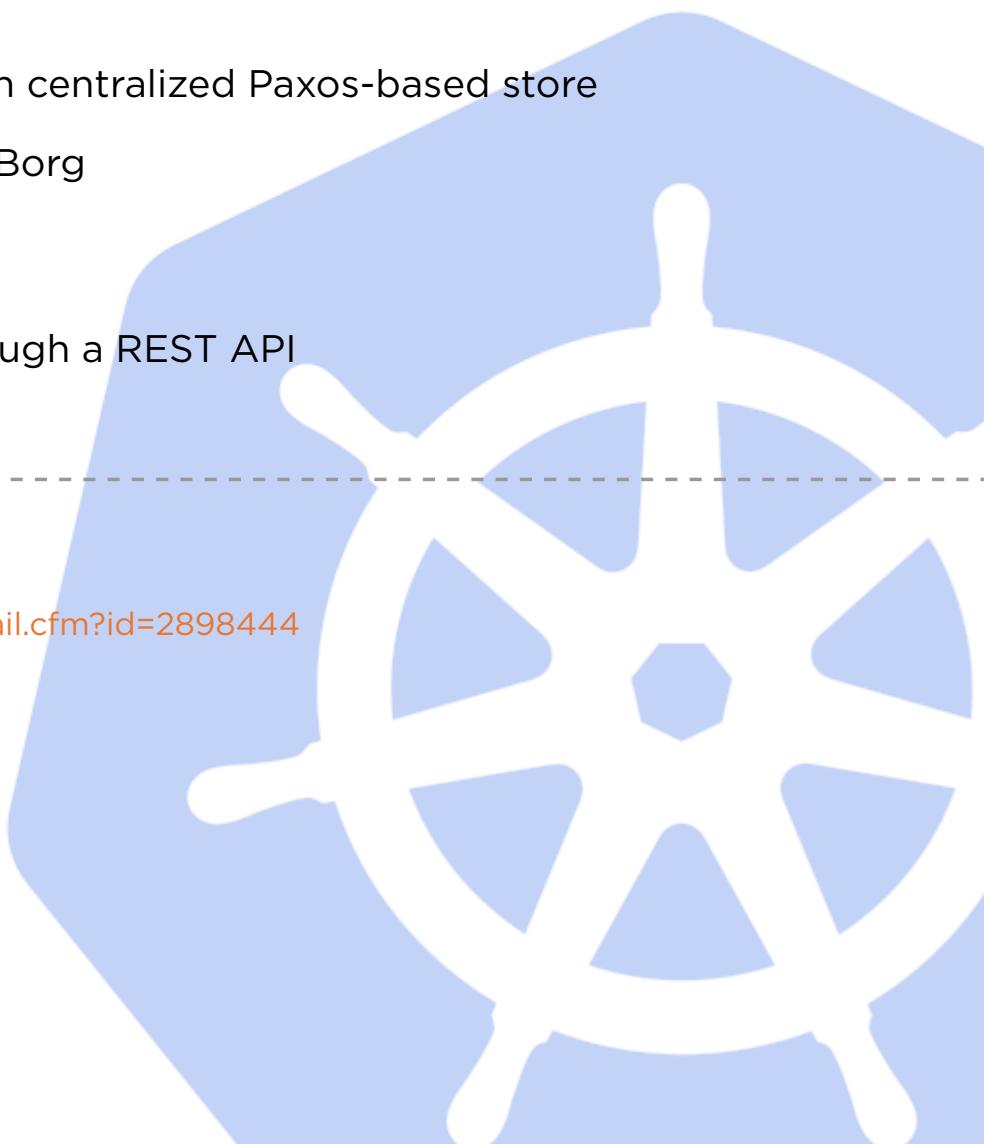
- Cluster state stored in centralized Paxos-based store
- Contributed back to Borg

Kubernetes

- Everything goes through a REST API
- Open-Source!

Notes:

See <https://queue.acm.org/detail.cfm?id=2898444>



CNCF

- Cloud Native Computing Foundation



- Camptocamp is an official Partner



Checkpoint: Kubernetes Basics

- Which container engines are supported by Kubernetes?
 - Docker
 - rkt
 - cri-o
- Kubernetes is to Docker what Openstack is to...
 - Debian
 - Linux
 - kvm
 - AWS
- Which platforms are based on Kubernetes?
 - Amazon ECS
 - Rancher 1
 - Rancher 2
 - Amazon EKS



KUBERNETES UIS

Lesson 18: Web Interfaces

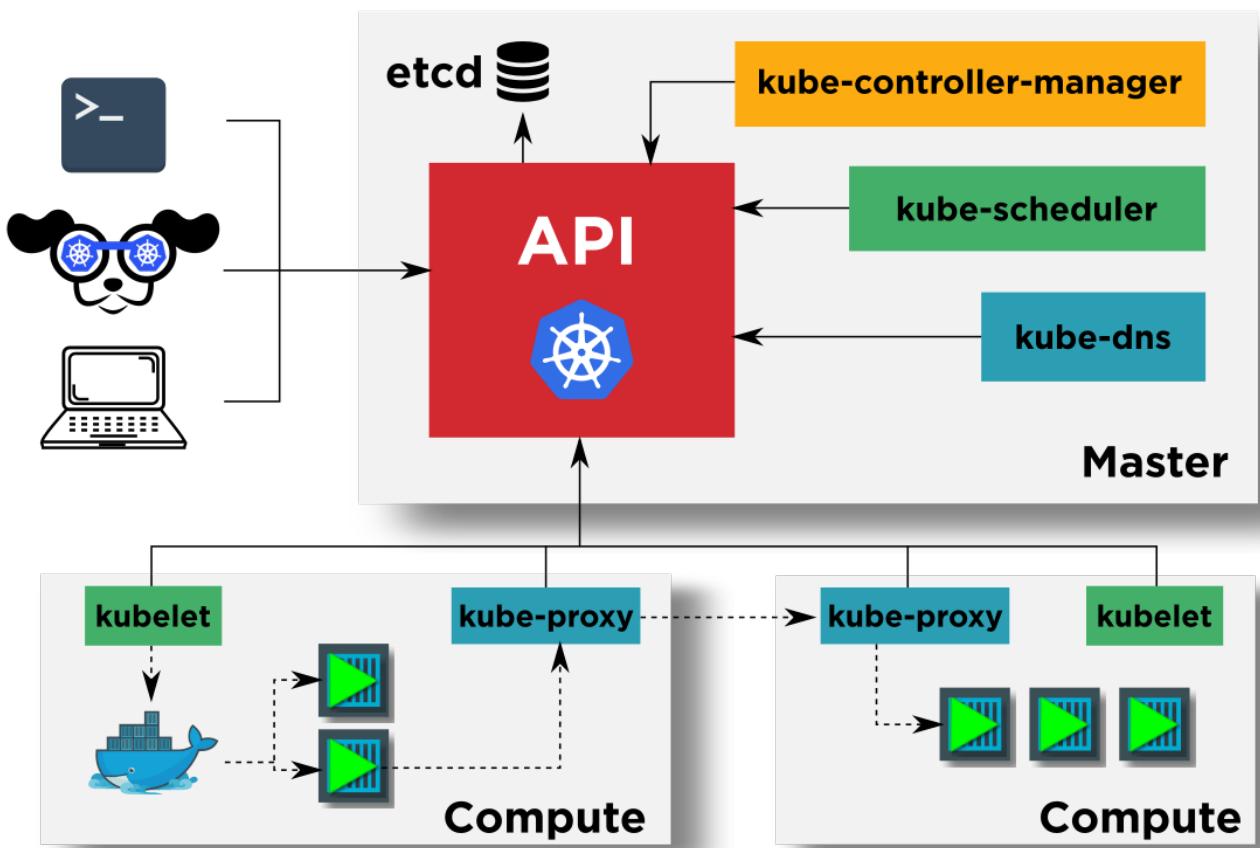
Objectives

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

- Know about the various ways to control Kubernetes objects
- Know about the various Web User Interfaces for Kubernetes
- Understand the difference between a Kubernetes control panels and Openshift



Platform Overview



Platform Installation



- On Amazon EKS
- Using Terraform



K9S



K9S is a lightweight client UI demo

Context:	minikube	<?>	Help	<0>	all
Cluster:	minikube	<ctrl-d>	Delete	<1>	kube-system
User:	minikube	<d>	Describe	<2>	default
K9s Version:	0.1.6	<e>	Edit		
K8s Version:	v1.13.2	<l>	Logs		
CPU:	10%(-)	<s>	Shell		
MEM:	20%(+)	<v>	View		

Pods(all)[12]										
NAMESPACE	NAME	READY	STATUS	RESTARTS	CPU	MEM	IP	NODE	QOS	AGE
default	nginx-6988c9989f-wwz6d	1/1	Running	0			172.17.0.6	192.168.64.83	Guaranteed	87s
kube-system	coredns-86c58d9df4-dkhf2	1/1	Running	0	3m	8Mi	172.17.0.3	192.168.64.83	Burstable	17h
kube-system	coredns-86c58d9df4-jt79s	1/1	Running	0	2m(-)	8Mi	172.17.0.2	192.168.64.83	Burstable	17h
kube-system	etcd-minikube	1/1	Running	0	24m(-)	53Mi	192.168.64.83	192.168.64.83	BestEffort	17h
kube-system	kube-addon-manager-minikube	1/1	Running	0	7m(+)	17Mi(-)	192.168.64.83	192.168.64.83	Burstable	17h
kube-system	kube-apiserver-minikube	1/1	Running	0	47m(-)	383Mi	192.168.64.83	192.168.64.83	Burstable	17h
kube-system	kube-controller-manager-minikube	1/1	Running	0	49m(-)	55Mi(+)	192.168.64.83	192.168.64.83	Burstable	17h
kube-system	kube-proxy-pjh2p	1/1	Running	0	4m(-)	10Mi	192.168.64.83	192.168.64.83	BestEffort	17h
kube-system	kube-scheduler-minikube	1/1	Running	0	15m(-)	12Mi	192.168.64.83	192.168.64.83	Burstable	17h
kube-system	kubernetes-dashboard-ccc79bfc9-qgnck	1/1	Running	0	0m	11Mi	172.17.0.4	192.168.64.83	BestEffort	17h
kube-system	metrics-server-6fc4b7bcff-hp6pm	1/1	Running	0	1m	14Mi	172.17.0.5	192.168.64.83	BestEffort	17h
kube-system	storage-provisioner	1/1	Running	0	0m	13Mi	192.168.64.83	192.168.64.83	BestEffort	17h

K9S – Usage

Can be run in read-only mode : `k9s --readonly`

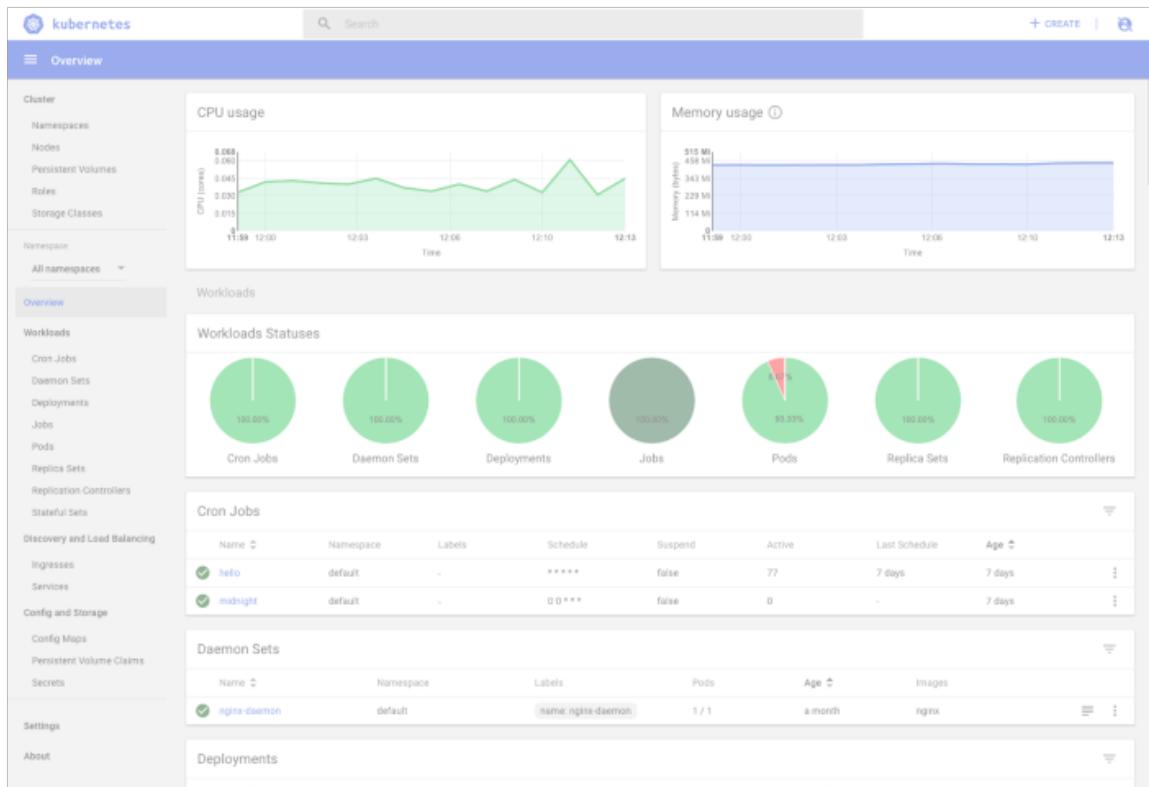
- `:ns` List Namespaces
- `:pod`, `:deploy`, `:service` List Pods, Deployments or Services.
- `Enter`, `Esc` Navigate from Deployments to Pods.
- `ctrl+s` Save to temporary file. CSV for table or YAML.
- `:pulses`, `:xray deployment`, `:popeye` Some nice overview
- `^A`: list all known commands and objects
- `:q` Quit K9s

Deployment and *Pod* logs are aggregated



Kubernetes Dashboard

- Native Web UI for Kubernetes Clusters
- Manage and troubleshoot applications
- Manage cluster
- Run inside cluster as *Deployment*



Kubernetes Dashboard

localhost:9090/#/pod?namespace=kube-system

kubernetes Workloads > Pods + CREATE

Admin

- Namespaces
- Nodes
- Persistent Volumes

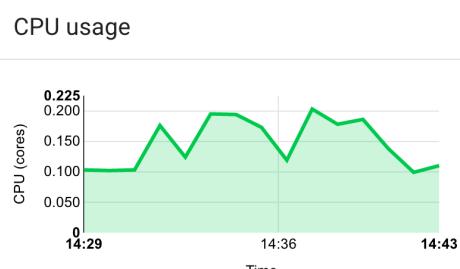
Namespace

- kube-system** ▾

Workloads

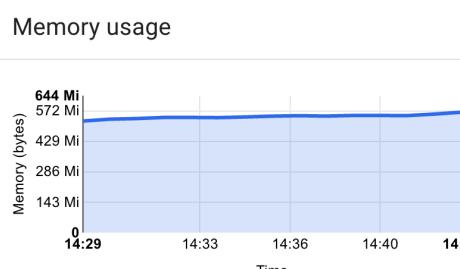
- Deployments
- Replica Sets
- Replication Controllers
- Daemon Sets
- Stateful Sets
- Jobs
- Pods**
- Services and discovery
- Services

CPU usage



Time: 14:29, 14:36, 14:43

Memory usage

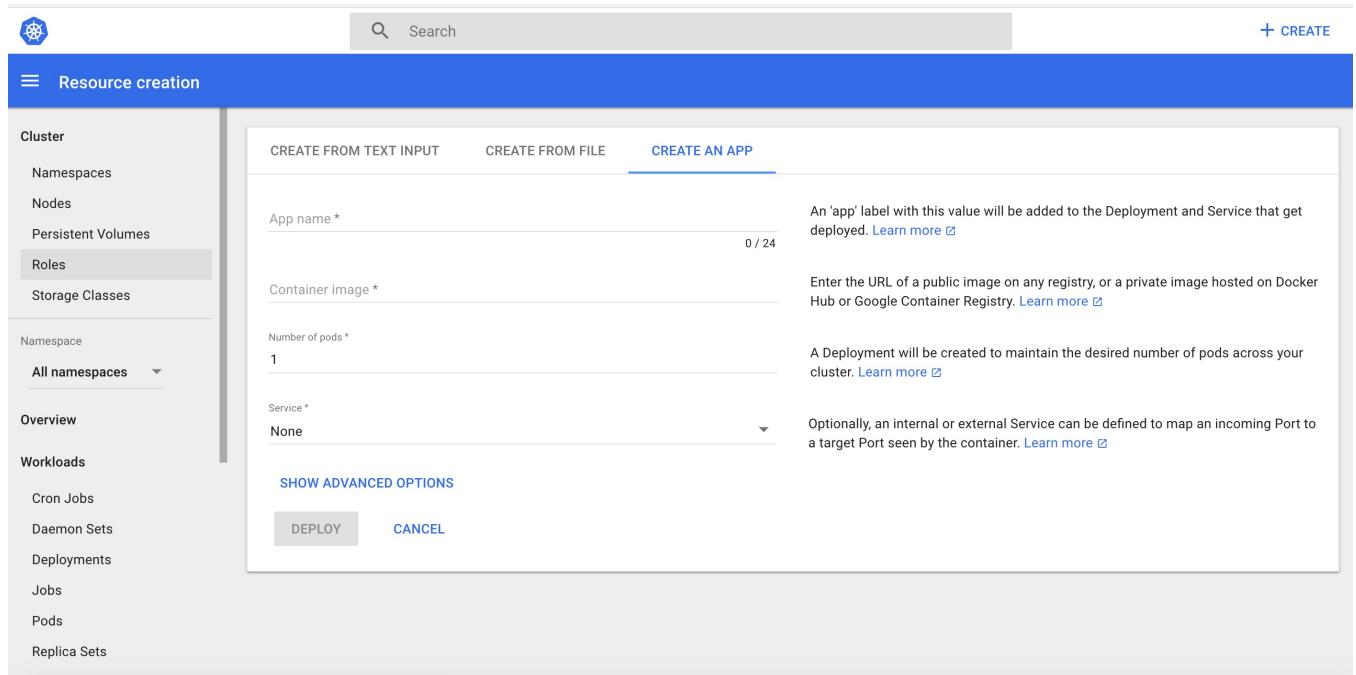


Time: 14:29, 14:33, 14:36, 14:40, 14:43

Pods

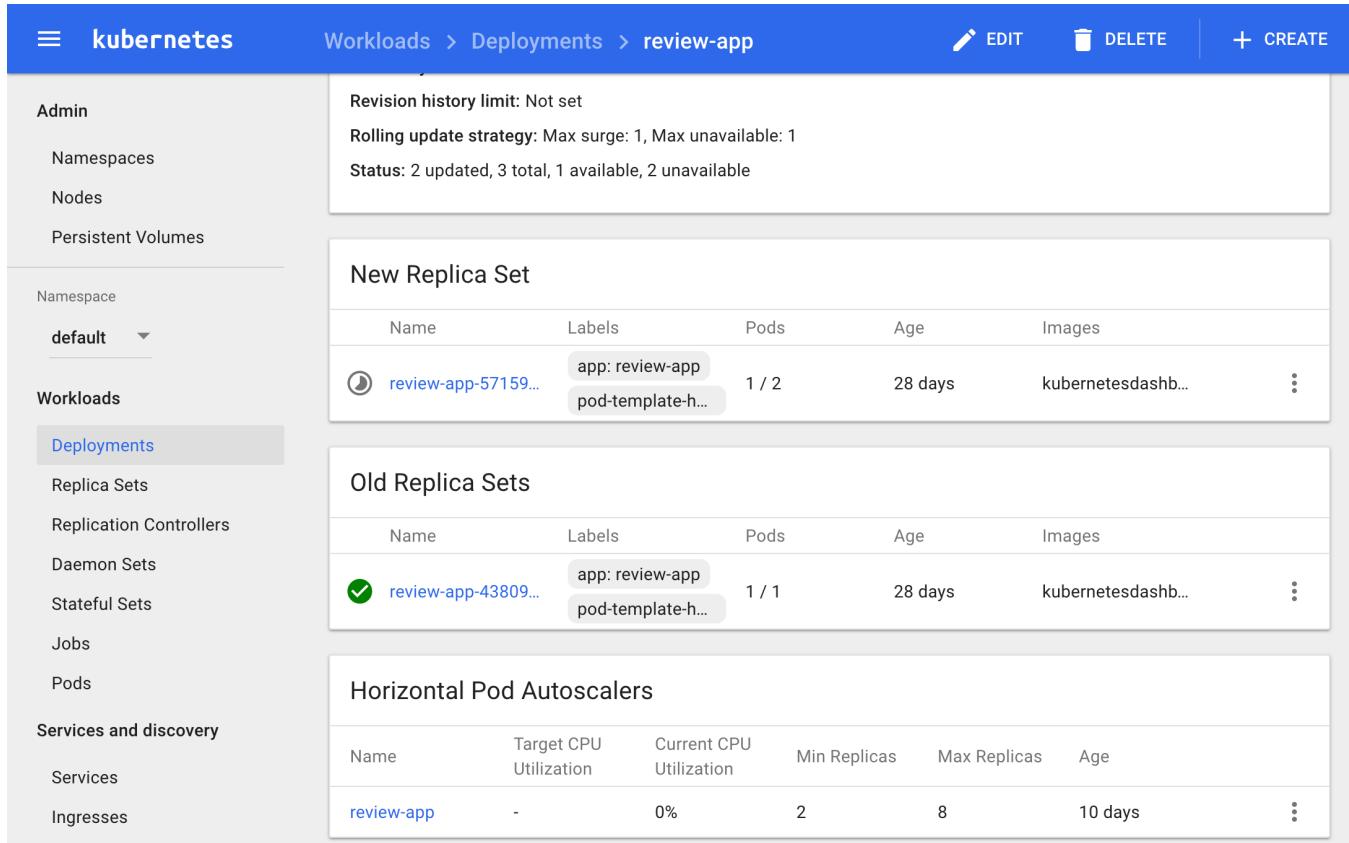
Name	Status	Restarts	Age	CPU (cores)	Memory (bytes)	⋮
dashboard-same-i...	Running	0	14 minutes	0	5.4 Mi	⋮
fluentd-cloud-logg...	Running	0	2 months	0.01	120.8 Mi	⋮
fluentd-cloud-logg...	Running	0	2 months	0.009	94.3 Mi	⋮
fluentd-cloud-logg...	Running	0	2 months	0.014	174.3 Mi	⋮
heapster-v1.2.0-4...	Running	0	10 days	0.002	44.8 Mi	⋮

Kubernetes Dashboard - App Creation



The screenshot shows the Kubernetes Dashboard interface for creating a new application. The left sidebar is titled 'Resource creation' and includes sections for Cluster (Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes), Namespace (All namespaces dropdown), Overview, and Workloads (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets). The main content area has a blue header bar with tabs: 'CREATE FROM TEXT INPUT', 'CREATE FROM FILE', and 'CREATE AN APP' (which is selected). Below the tabs are input fields for 'App name *' (with a character count of 0 / 24), 'Container image *', 'Number of pods *' (set to 1), and 'Service *' (set to None). There is also a 'SHOW ADVANCED OPTIONS' link. At the bottom are 'DEPLOY' and 'CANCEL' buttons. To the right of the input fields are explanatory notes: one for the app name (mentioning it will be added to Deployment and Service labels), one for the container image (mentioning Docker Hub or Google Container Registry), one for the number of pods (mentioning a Deployment will be created), and one for the service (mentioning port mapping).

Kubernetes Dashboard – Update/Scale



The screenshot shows the Kubernetes Dashboard interface for the `review-app` deployment. The left sidebar is collapsed, and the main navigation bar shows `Workloads > Deployments > review-app`. The top right has buttons for `EDIT`, `DELETE`, and `+ CREATE`.

New Replica Set:

Name	Labels	Pods	Age	Images
review-app-57159...	app: review-app pod-template-h...	1 / 2	28 days	kubernetesdashb...

Old Replica Sets:

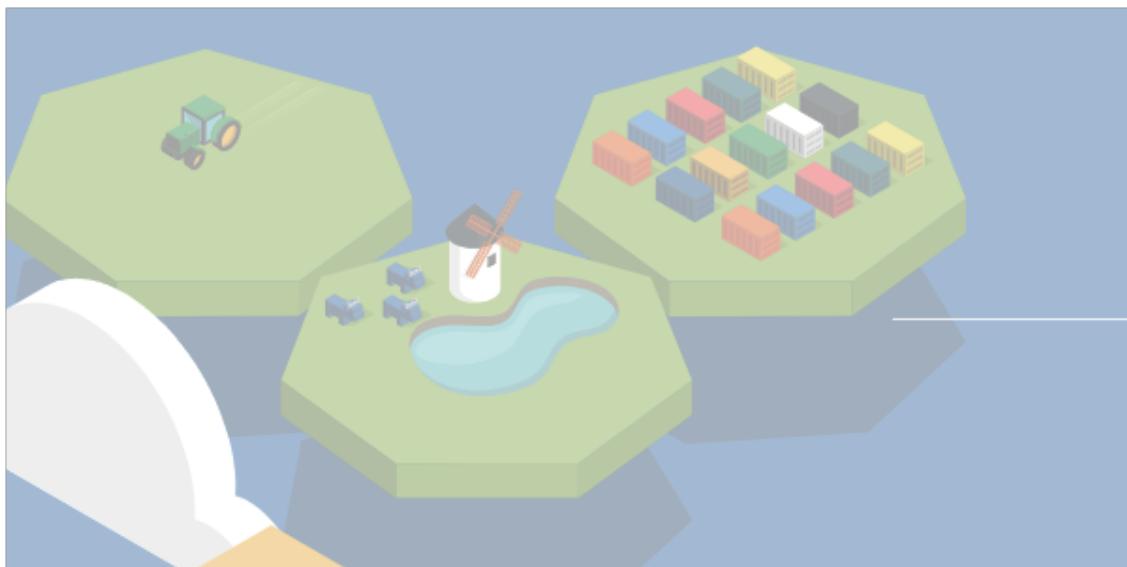
Name	Labels	Pods	Age	Images
review-app-43809...	app: review-app pod-template-h...	1 / 1	28 days	kubernetesdashb...

Horizontal Pod Autoscalers:

Name	Target CPU Utilization	Current CPU Utilization	Min Replicas	Max Replicas	Age
review-app	-	0%	2	8	10 days

Rancher 2

- From version 2, Rancher uses Kubernetes as orchestrator
- Includes Jenkins pipeline CI
- Aggregate logs and send it to Elasticsearch, Kafka, Syslog, ...
- Compatible with the Helm Kubernetes package manager



Rancher 2 - Node Drivers

The screenshot shows the Rancher 2 interface for managing Node Drivers. At the top, there's a navigation bar with icons for Global, Clusters, Node Drivers (which is the active tab), Catalogs, Users, Settings, and Security. On the right of the bar is a cluster selection dropdown. Below the bar, a header bar has buttons for Activate, Deactivate, Delete, and Add Node Driver. A search bar is also present. The main area displays a table of node drivers, sorted by State (Inactive by default). The table includes columns for State (with Active and Inactive buttons), Name, and URL. Each row has a three-dot menu icon on the far right.

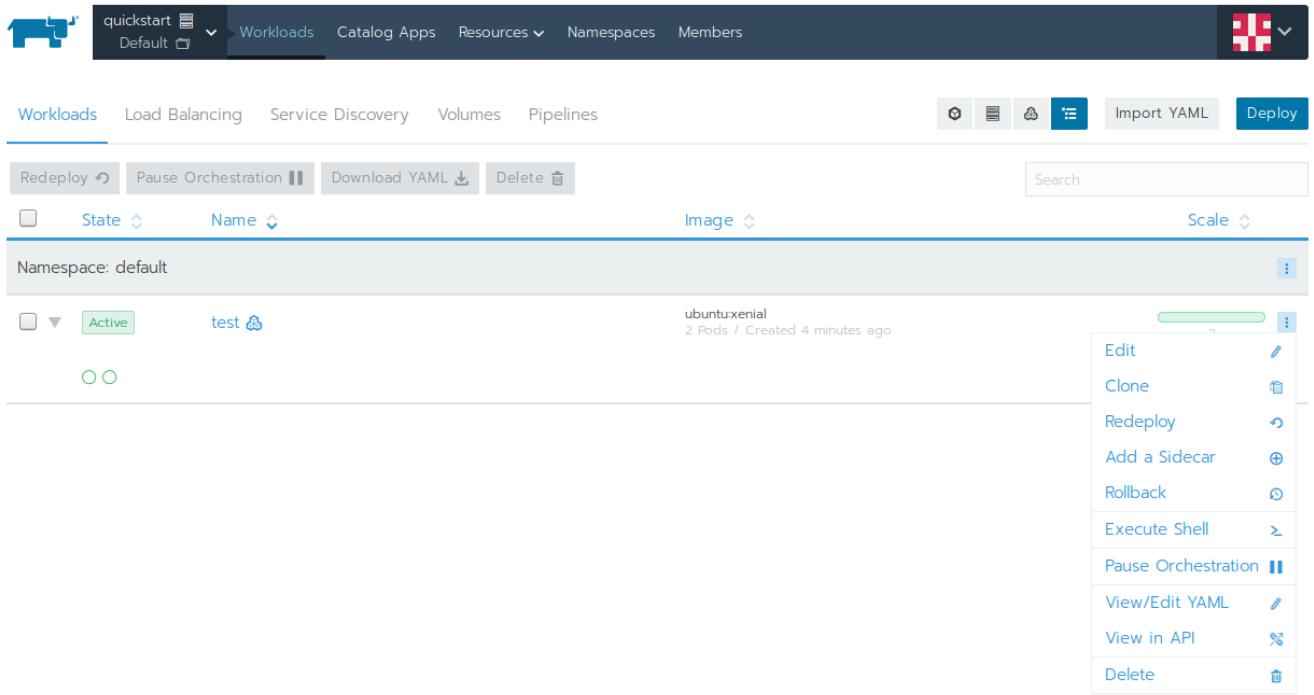
State	Name	URL
Inactive	Aliyun ECS	http://machine-driver.oss-cn-shanghai.aliyuncs.com/aliyun/10.2/linux/amd64/docker-machine-driver-aliyuncs.tgz
Active	Amazon EC2	Built-In
Active	Azure	Built-In
Active	DigitalOcean	Built-In
Inactive	Exoscale	Built-In
Inactive	OpenStack	Built-In
Inactive	Open Telekom Cloud	https://dockermachinedriver.obs.eu-de.otc.t-systems.com/docker-machine-driver-otc
Inactive	Packet	https://github.com/packethost/docker-machine-driver-packet/releases/download/v0.14/docker-machine-driver-packet_linux-amd64.zip
Inactive	RackSpace	Built-In
Inactive	SoftLayer	Built-In
Active	vSphere	Built-In

Rancher 2 - Catalog

The screenshot shows the Rancher 2 Catalog interface. At the top, there's a navigation bar with tabs for 'quickstart' (selected), 'Workloads', 'Catalog Apps', 'Resources', 'Namespaces', 'Members', and a search bar. Below the navigation is a search bar with dropdowns for 'All Categories' and 'All Catalogs'. The main area displays a grid of application charts:

- artifactory-ha (from Library)**: Universal Repository Manager supporting all major packaging formats, build tools and CI servers. [View Details](#)
- Let's Encrypt cert-manager (from Library)**: A Helm chart for cert-manager. [View Details](#)
- chartmuseum (from Library)**: Helm Chart Repository with support for Amazon S3 and Google Cloud Storage. [View Details](#)
- datadog (from Library)**: Datadog Agent. [View Details](#)
- docker-registry (from Library)**: A Helm chart for Docker Registry. [View Details](#)
- drone (from Library)**: Drone is a Continuous Delivery system built on container technology. [View Details](#)
- efk (from Library)**: EFK(Elasticsearch + FluentBit + Kibana). [View Details](#)
- etcd-operator (from Library)**: CoreOS etcd-operator Helm chart for Kubernetes. [View Details](#)
- grafana (from Library)**: Grafana instance for kube-prometheus. [View Details](#)
- hadoop (from Library)**: The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers, using simple, reliable map/reduce programs. [View Details](#)
- istio (from Library)**: Helm chart for all istio components. [View Details](#)
- kafka (from Library)**: Apache Kafka is publish-subscribe messaging rethought as a distributed commit log. [View Details](#)

Rancher 2 – Workload



The screenshot shows the Rancher 2 interface for managing workloads. At the top, there's a navigation bar with links for 'quickstart', 'Workloads' (which is selected), 'Catalog Apps', 'Resources', 'Namespaces', and 'Members'. On the right of the navigation bar are icons for 'Import YAML' and 'Deploy'. Below the navigation bar is a toolbar with buttons for 'Redeploy', 'Pause Orchestration', 'Download YAML', and 'Delete'. There's also a 'Search' input field.

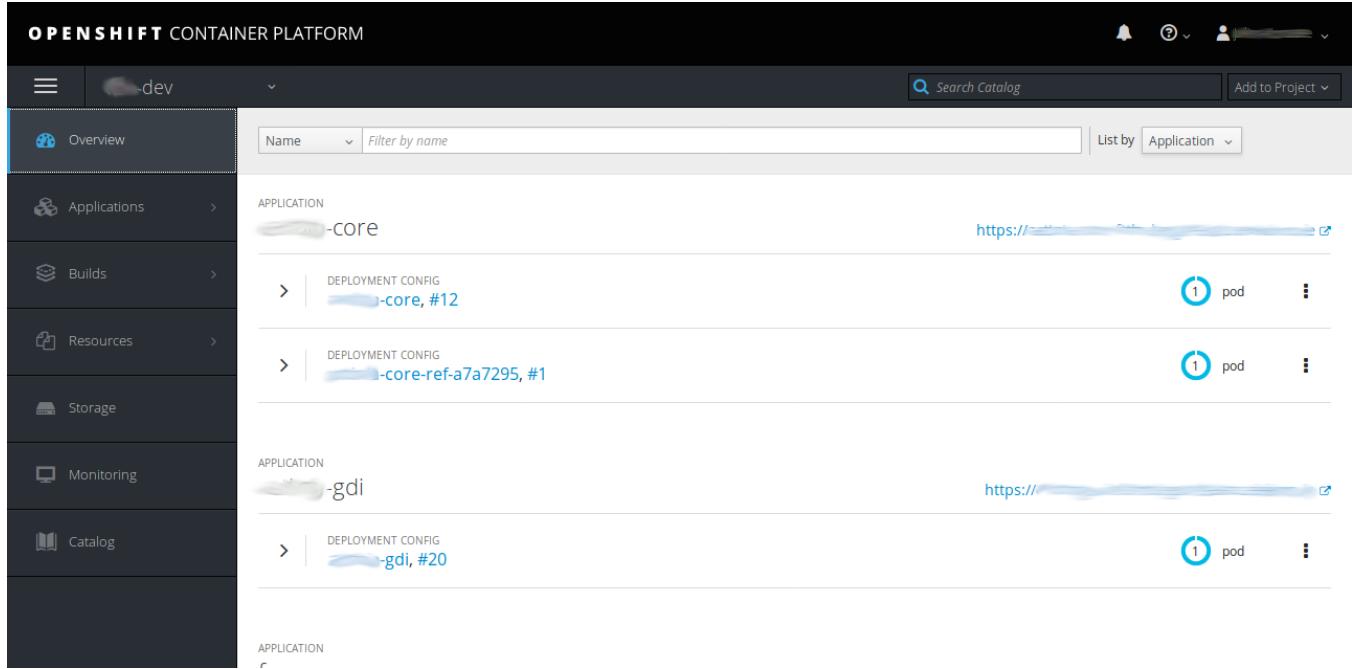
The main area displays a table of workloads. The columns are 'State' (with dropdown arrows), 'Name' (with dropdown arrows), 'Image' (with dropdown arrow), and 'Scale' (with dropdown arrow). A filter 'Namespace: default' is applied. One workload is listed: 'test' (Status: Active, Image: ubuntu xenial, 2 Pods / Created 4 minutes ago). To the right of this entry is a context menu with options: Edit, Clone, Redeploy, Add a Sidecar, Rollback, Execute Shell, Pause Orchestration, View/Edit YAML, View in API, and Delete.

Openshift

- RedHat solution for container platform
- Provides PaaS or dedicated instance
- Several versions:
 - local single host version: Minishift
 - Open-Source version: OKD
 - full version with Red Hat support: OpenShift container platform



Openshift – Overview



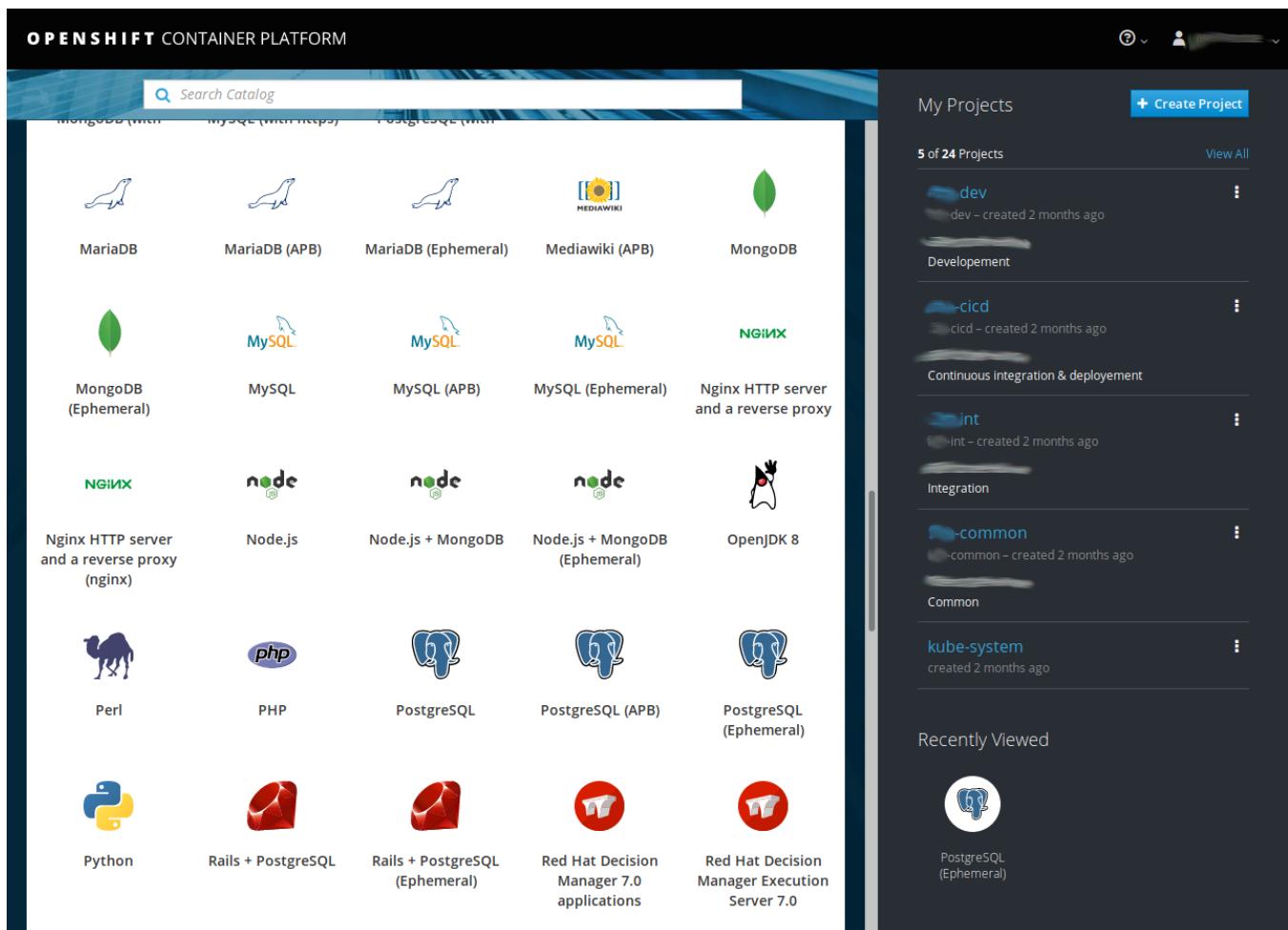
The screenshot shows the OpenShift Container Platform interface. The left sidebar includes links for Overview, Applications, Builds, Resources, Storage, Monitoring, and Catalog. The main area displays two applications:

- core**: DEPLOYMENT CONFIG -core, #12 (https://...), 1 pod
- gdi**: DEPLOYMENT CONFIG -gdi, #20 (https://...), 1 pod

Openshift – Pod

The screenshot shows the OpenShift Container Platform interface. The left sidebar includes options like Overview, Applications, Builds, Resources, Storage, Monitoring, and Catalog. The main content area displays a pod named "pghoard-pghoard-1-mtqhp" under the "cluster-backup" project. The pod was created a month ago and is currently running. It is associated with a deployment named "pghoard-pghoard, #1". The pod has an IP address of 10.129.5.43 and is running on node 10.171.185.60. The restart policy is set to Always. The container "pghoard" is running since Oct 12, 2018, at 11:59:52 AM, but terminated at 11:59:49 AM with exit code 137 (Error). The container's ready status is true, and it has a restart count of 1. The right side of the screen shows the pod's template, which includes an image (camptocamp/pghoard), command (pghoard --config /etc/pghoard/pghoard.json), and several mounts. It also lists volumes: config-volume (config map), pgoard (persistent volume claim), and default-token-x4stl (secret). Buttons for "Add Storage to pgoard-pgoard" and "Add Config Files to pgoard-pgoard" are visible.

Openshift – Catalog



The screenshot shows the OpenShift Container Platform Catalog interface. At the top, there is a search bar labeled "Search Catalog". Below the search bar, there are several project templates displayed in a grid:

- MariaDB
- MariaDB (APB)
- MariaDB (Ephemeral)
- Mediawiki (APB)
- MongoDB
- MongoDB (Ephemeral)
- MySQL
- MySQL (APB)
- MySQL (Ephemeral)
- NGINX
- NGINX HTTP server and a reverse proxy (nginx)
- node
- Node.js
- Node.js + MongoDB
- Node.js + MongoDB (Ephemeral)
- OpenJDK 8
- Perl
- PHP
- PostgreSQL
- PostgreSQL (APB)
- PostgreSQL (Ephemeral)
- Python
- Rails + PostgreSQL
- Rails + PostgreSQL (Ephemeral)
- Red Hat Decision Manager 7.0 applications
- Red Hat Decision Manager Execution Server 7.0

On the right side, there is a sidebar titled "My Projects" which lists 5 of 24 projects:

- dev - created 2 months ago
- Development
- cicd - created 2 months ago
- Continuous integration & deployment
- lint - created 2 months ago
- Integration
- common - common - created 2 months ago
- Common
- kube-system
- created 2 months ago

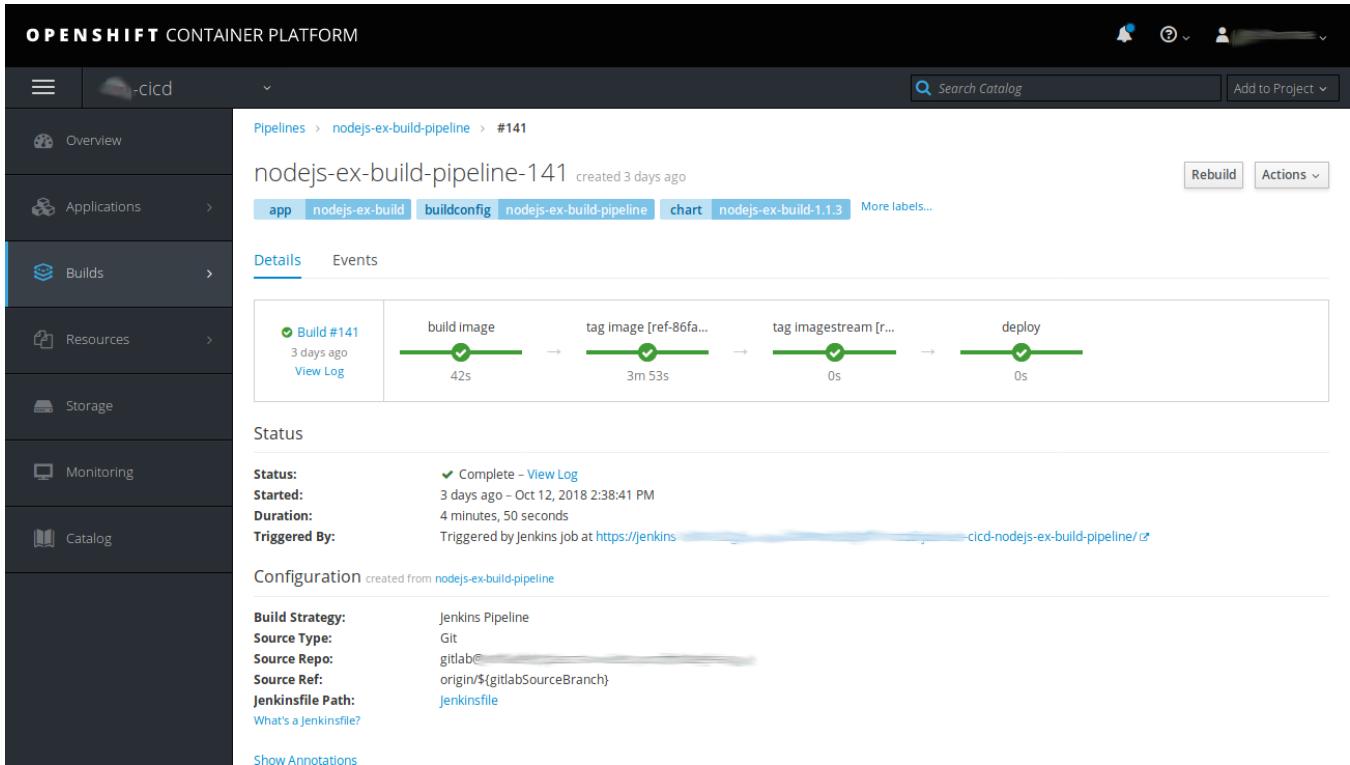
Below the project list, there is a "Recently Viewed" section showing a thumbnail for "PostgreSQL (Ephemeral)".

Openshift – Pipeline build

The screenshot shows the OpenShift Container Platform interface with the following details:

- Left Sidebar:** Includes sections for Overview, Applications, Builds (selected), Resources, Storage, Monitoring, and Catalog.
- Top Bar:** Shows the project name "-ci-cd", search bar, and user profile.
- Pipeline Runs:**
 - pipeline**: Created 4 days ago. Recent Run: Build #56 (3 hours ago). Stages: build image (48m 16s), build child images (1h 30m), tag child images [r...], tag child imagestream [r...], deploy (5s). Average Duration: 13m 41s.
 - nodejs-ex-build-pipeline**: Created 13 days ago. Recent Run: Build #141 (3 days ago). Stages: build image (42s), tag image [ref-86f...], tag imagestream [...], deploy (0s). Average Duration: 1m 54s.
 - Build #128: No stages have started.
- Bottom Footer:** © Camptocamp 2021 / V1.3.0 / 04.06.2021, KUBERNETES UIS, 329/579

Openshift – Pipeline build



The screenshot shows the OpenShift Container Platform interface. The left sidebar has a dark theme with the following navigation items:

- Overview
- Applications
- Builds
- Resources
- Storage
- Monitoring
- Catalog

The main content area displays a pipeline build named "nodejs-ex-build-pipeline-141" created 3 days ago. The pipeline consists of four steps: "build image", "tag image [ref-86fa...]", "tag imagestream [r...]", and "deploy". All steps are marked as "Complete" with green checkmarks. The "build image" step took 42s, "tag image" took 3m 53s, "tag imagestream" took 0s, and "deploy" took 0s.

Status: ✓ Complete – [View Log](#)
Started: 3 days ago – Oct 12, 2018 2:38:41 PM
Duration: 4 minutes, 50 seconds
Triggered By: Triggered by Jenkins job at <https://jenkins/cicd-nodejs-ex-build-pipeline/>

Configuration (created from nodejs-ex-build-pipeline)

Build Strategy:	Jenkins Pipeline
Source Type:	Git
Source Repo:	gitlab@[REDACTED]
Source Ref:	origin/\${gitlabSourceBranch}
Jenkinsfile Path:	Jenkinsfile

[What's a Jenkinsfile?](#)

[Show Annotations](#)

Openshift – Pipeline build

✓  -cicd /  cicd/nodejs-ex-build-pipeline < 148

Branch: -  4m 50s No changes
Commit: -  3 days ago Started by GitLab push by

Pipeline Changes Tests Artifacts ⌂ ⚙️ ↻ Logout X

Description OpenShift Build  -cicd/nodejs-ex-build-pipeline- from gitlab@ nodejs-ex.git



```
graph LR; Start((Start)) --> buildImage((build image)); buildImage --> tagImage((tag image [ref-86fa369])); tagImage --> tagImageStream((tag imagestream [ref-86fa369])); tagImageStream --> deploy((deploy)); deploy --> End((End))
```

deploy - <1s  

✓ > Print Message <1s

Checkpoint: Web Interfaces

■ The Kubernetes Dashboard allows to

- Visualize Namespaces
- Visualize Nodes
- Visualize Pod logs
- Execute a shell in a Pod
- Edit Kubernetes Objects
- Provision new cluster Nodes

■ The Rancher 2 interface allows to

- Visualize Namespaces
- Visualize Nodes
- Visualize Pod logs
- Execute a shell in a Pod
- Edit Kubernetes Objects
- Provision new cluster Nodes

■ The OpenShift interface allows to

- Visualize Namespaces
- Visualize Nodes
- Visualize Pod logs
- Execute a shell in a Pod
- Edit Kubernetes Objects
- Provision new cluster Nodes

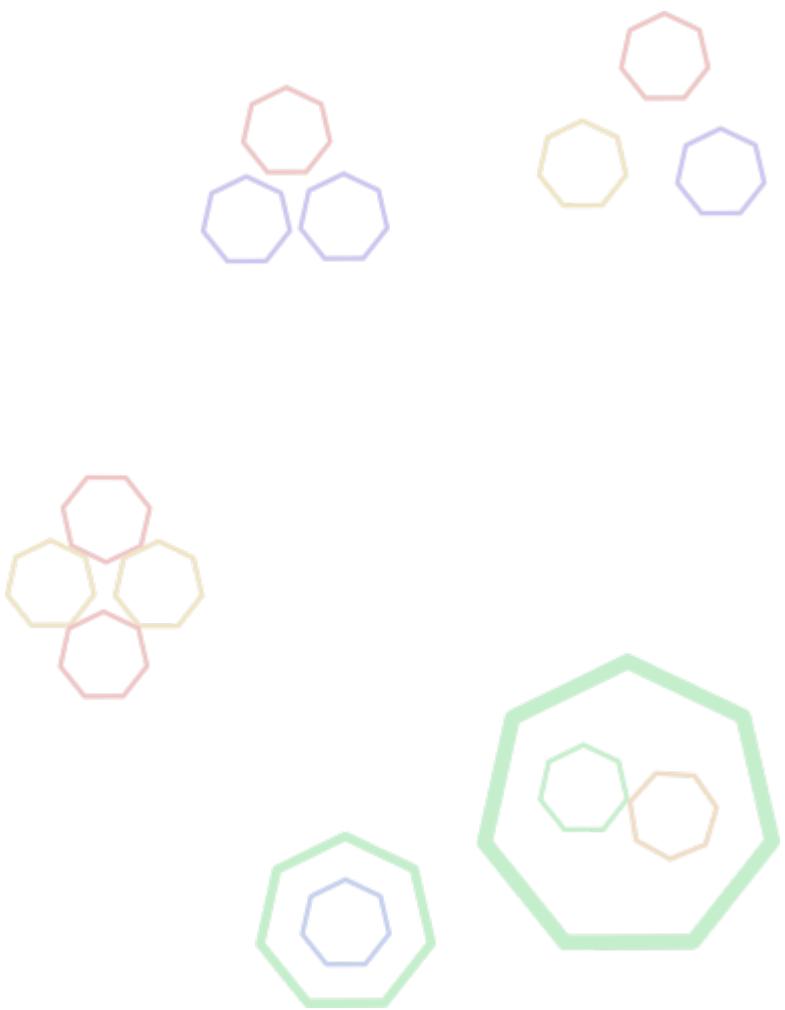
KUBERNETES CLI

Lesson 19: Command Line Interface

Objectives

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

- know the main functionalities of the `kubectl` command
- know the difference between imperative and declarative object workflows

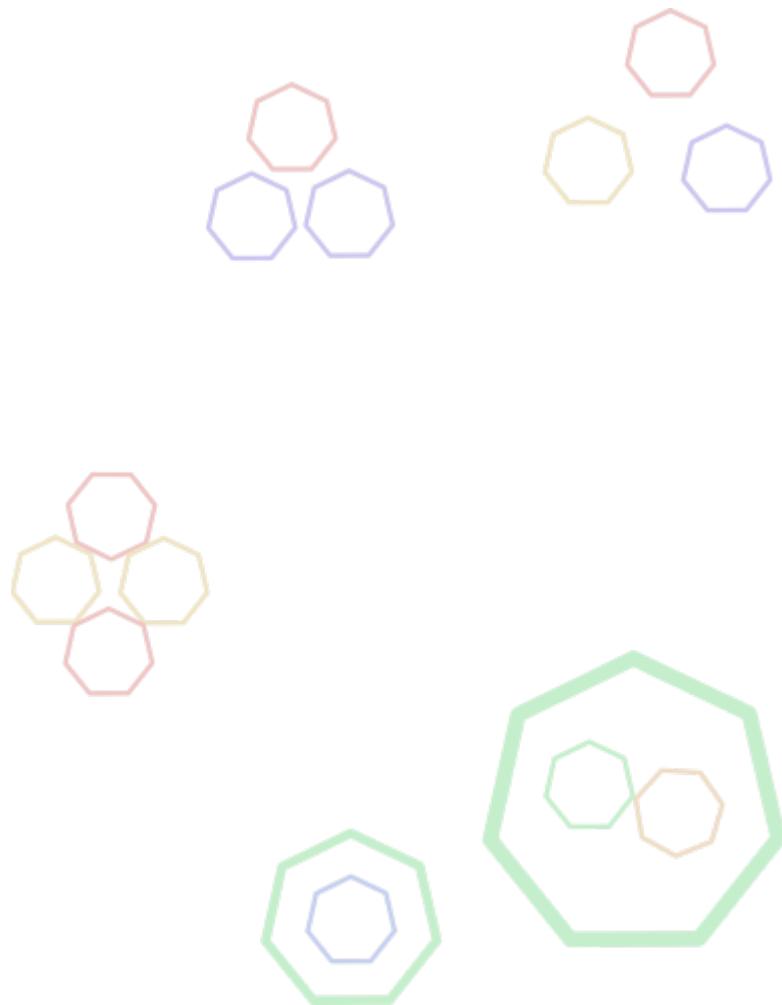


CLI with kubectl

Interact with the Kubernetes API:

- Create objects
- See object status
- Debug *Pods*
- Forward ports

```
$ kubectl get pods
$ kubectl logs webserver-1-dts34
```



Lab 19.1: Install and configure kubectl



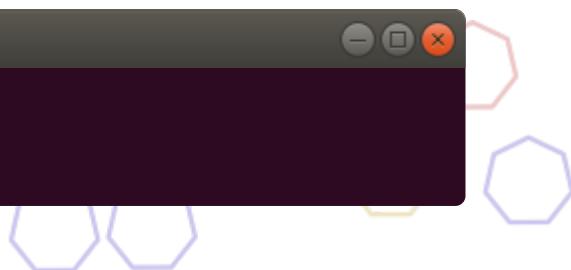
Objective

- Install and configure `kubectl`

Steps

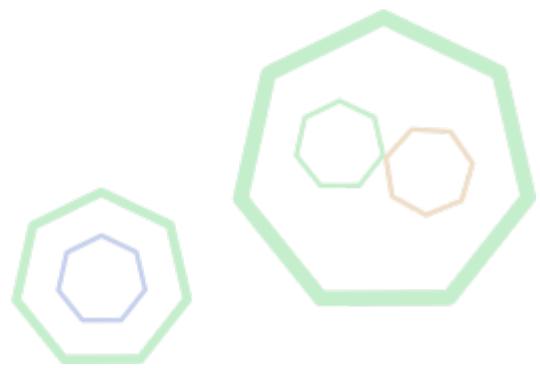
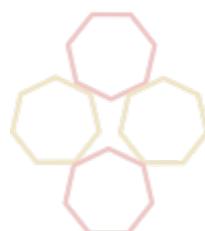
- Download and install the `kubectl` binary
- Make sure you have `curl` installed
- Download the `kubectl config` file from the course download section
- Install the `config` file in `~/.kube/`
- Verify that you can access the cluster:

```
$ kubectl config view  
$ kubectl get nodes  
$ kubectl get all
```



Questions

- How many Nodes are part of the cluster?
 - 1
 - 2
 - 3
 - more than 3



Lab 19.2: Inspect K9S



Objective

- Log in with K9S and inspect it

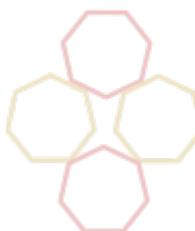
Steps

- Start k9s:

```
$ export KUBECONFIG=config  
$ k9s
```

Questions

- How many Pods are currently running ?
 - none
 - 1-4
 - 5-8
 - more than 8



Creating and deleting Kubernetes Objects

- Create an object from YAML file:

```
$ kubectl create -f FILENAME
```

- You can also pipe YAML definition from STDIN:

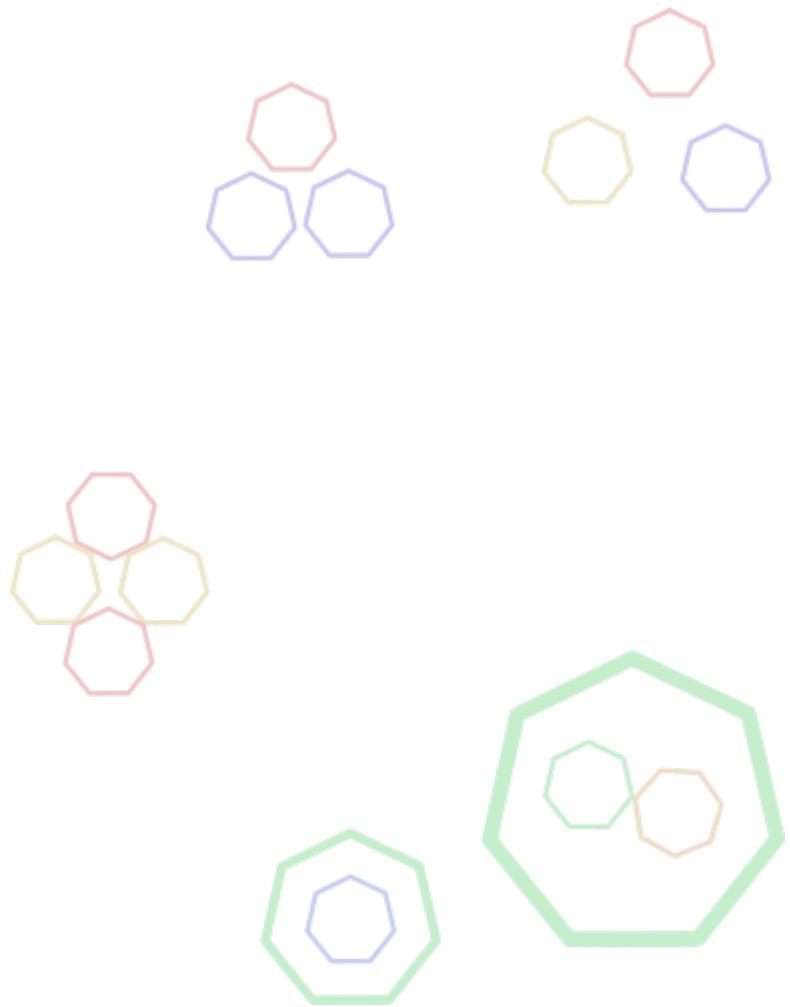
```
$ cat deployment.yaml | kubectl create -f -
```

- Delete an object:

```
$ kubectl delete <TYPE> <NAME>
```

References:

- [Kubectl Documentation](#)
- [Kubectl cheatsheet](#)



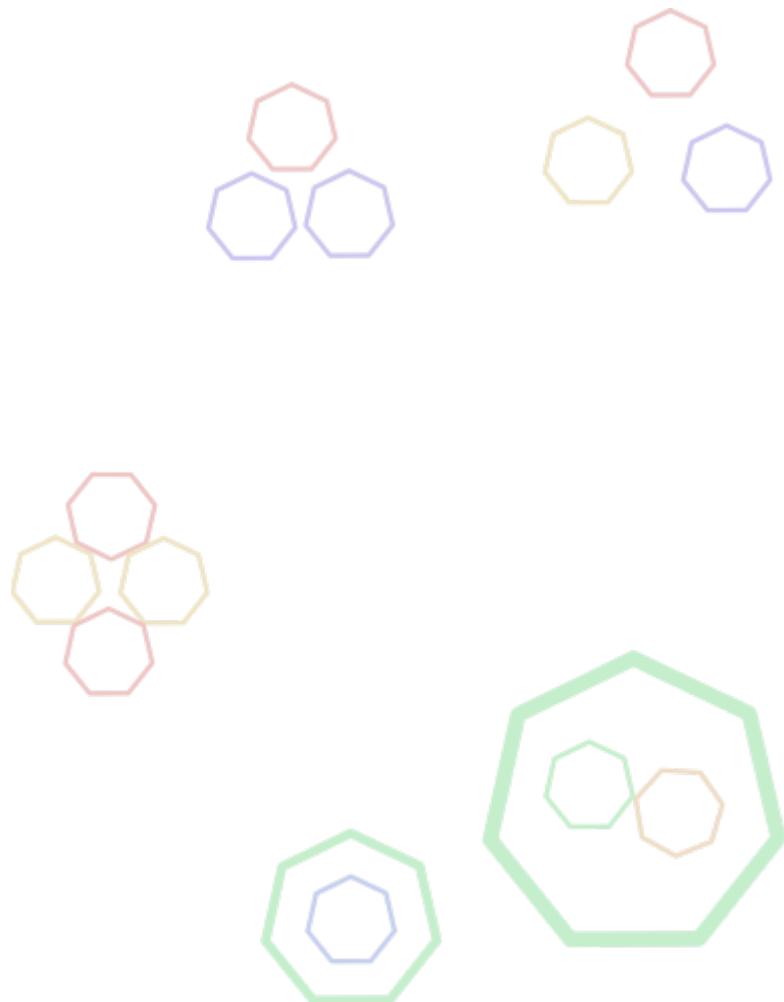
Json path

A way to extract value from object definition

Example: getting the `InternalIP` of all nodes

```
$ kubectl get nodes \
-o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}'
```

- See [JsonPath Reference](#)
- one can also use `jq` or `fx`



Updating Objects

- `kubectl edit TYPE NAME`: interactive edition with \$EDITOR:

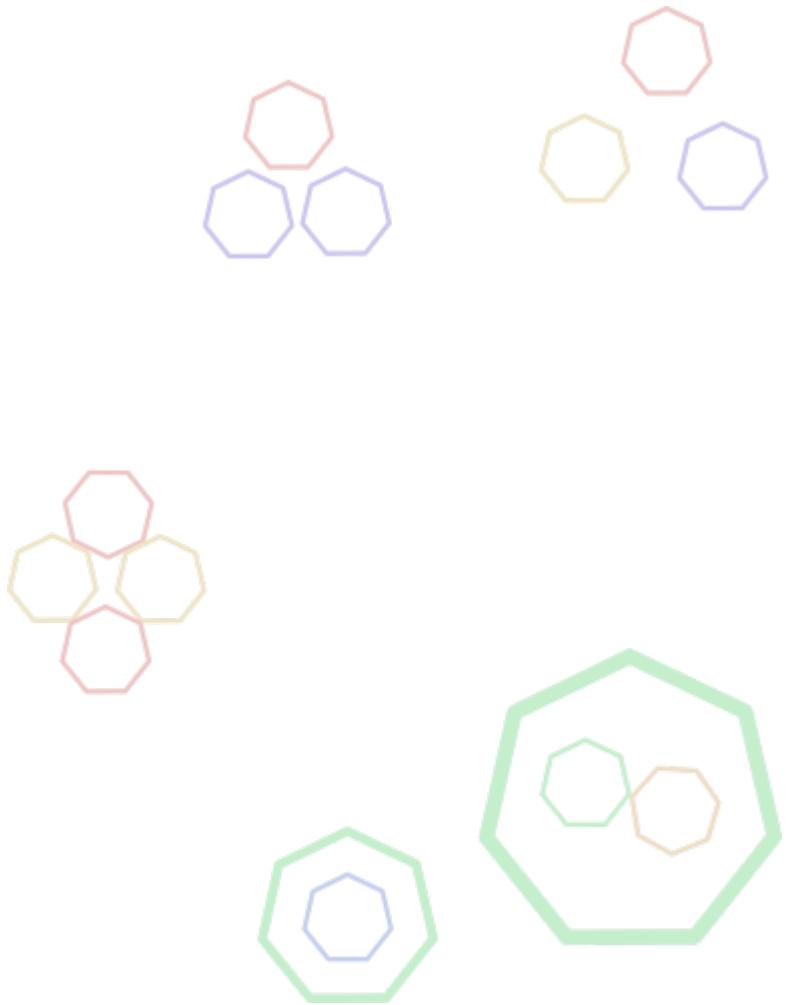
```
$ kubectl edit pod producer
```

- `kubectl replace -f FILENAME`: update object from file

```
$ kubectl replace -f deployment.yaml
```



You can use 'e' to edit object with k9s



The `kubectl create` command

- `create` allows the creation of objects based on:
 - command line arguments (image name, env vars, pod name)
 - JSON/YAML file
- Object must not already exist

```
$ kubectl create deployment --image=nginx
```

- Use `-o json --dry-run` options to generate a skeleton of pods, deployments, ...

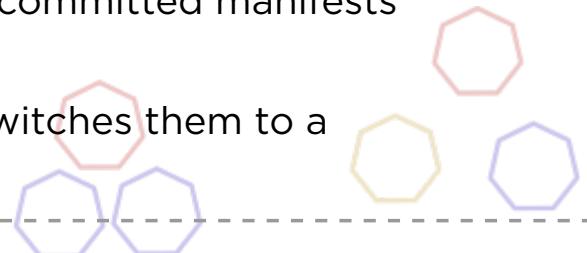
```
$ kubectl create service clusterip test --tcp=80:8080 -o json --dry-run
{
  "kind": "Service",
  "apiVersion": "v1",
  ...
}
```

Imperative vs Declarative

Two ways of managing objects with `kubectl`:

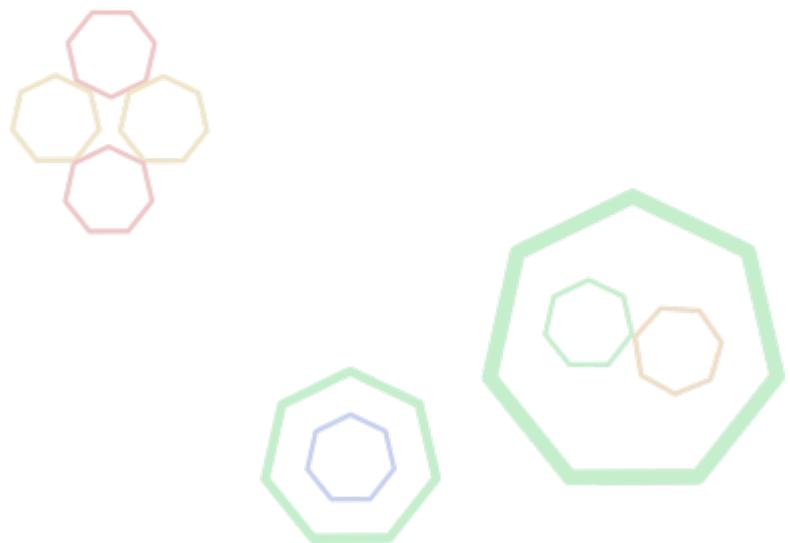
- Imperative: `create`, `edit / update / replace`, `delete`
 - `kubectl run` useful for Quick & Dirty tests, but limited in options
 - not idempotent
 - object conflicts
- Declarative: `apply`
 - converge towards a target state
 - no object conflicts
 - better approach on the long run, with committed manifests

Using `kubectl apply` on existing objects switches them to a declarative workflow.



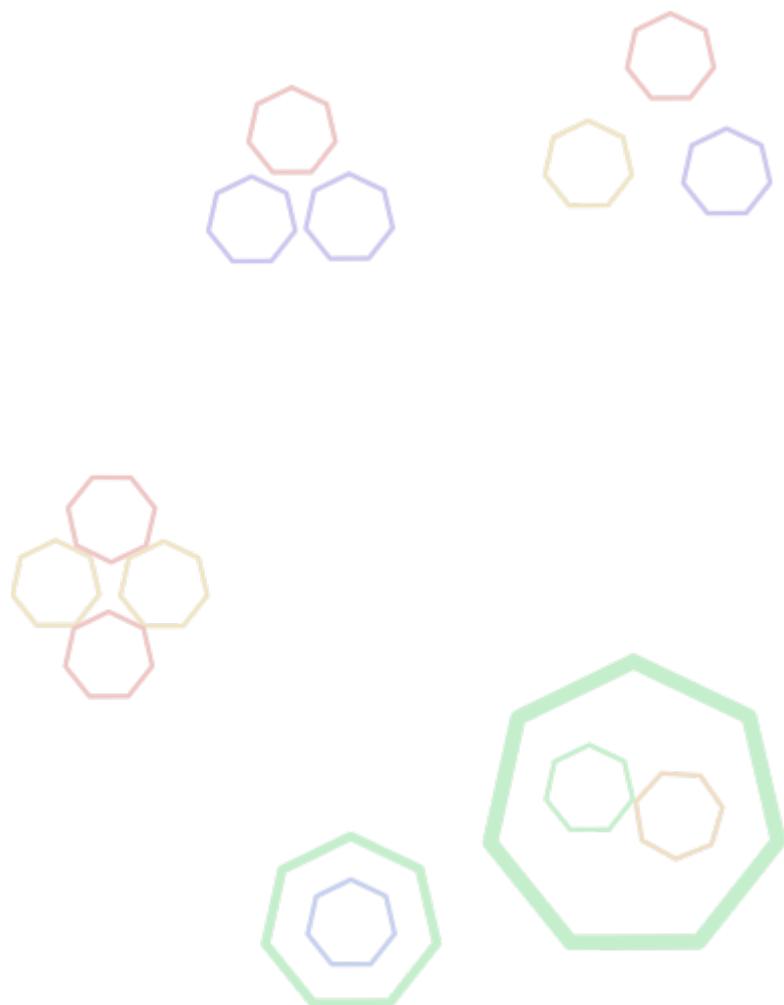
Notes:

- See <https://medium.com/bitnami-perspectives/imperative-declarative-and-a-few-kubectl-tricks-9d6deabde>



Declarative Workflow

- Use `kubectl apply -f <manifest>`
- Command converges to the desired state
- Some objects are immutable:
 - *Pod*
 - *Job*
- Usable in Continuous Integration



Listing objects

- `kubectl get <TYPE>` command lists objects in cluster
- `kubectl get <TYPE> <NAME>` provides a short description of one object
- some objects type:
 - *Pod, Cronjob, Job, Service*
 - *Deployment, Node, Namespace*
- Object types can be abbreviated, e.g.:
 - *po: Pod*
 - *no: Node*



You can list all object types and their shortcuts with `^A` in K9s



Listing Objects: output options

- `-o` changes the output format
 - `-o yaml` generates YAML format (object list or object definition)
 - `-o wide` adds details to object lists

```
$ kubectl get -o wide pods
NAME      READY   STATUS    RESTARTS   AGE      IP           NODE
consumer  1/1     Running   0          1d       10.50.0.222 ip-10-50-0-101.eu-
west-1.compute.internal
producer  1/1     Running   0          1d       10.50.0.180 ip-10-50-0-101.eu-
west-1.compute.internal
```

```
$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
consumer  1/1     Running   0          1d
producer  1/1     Running   0          1d
```



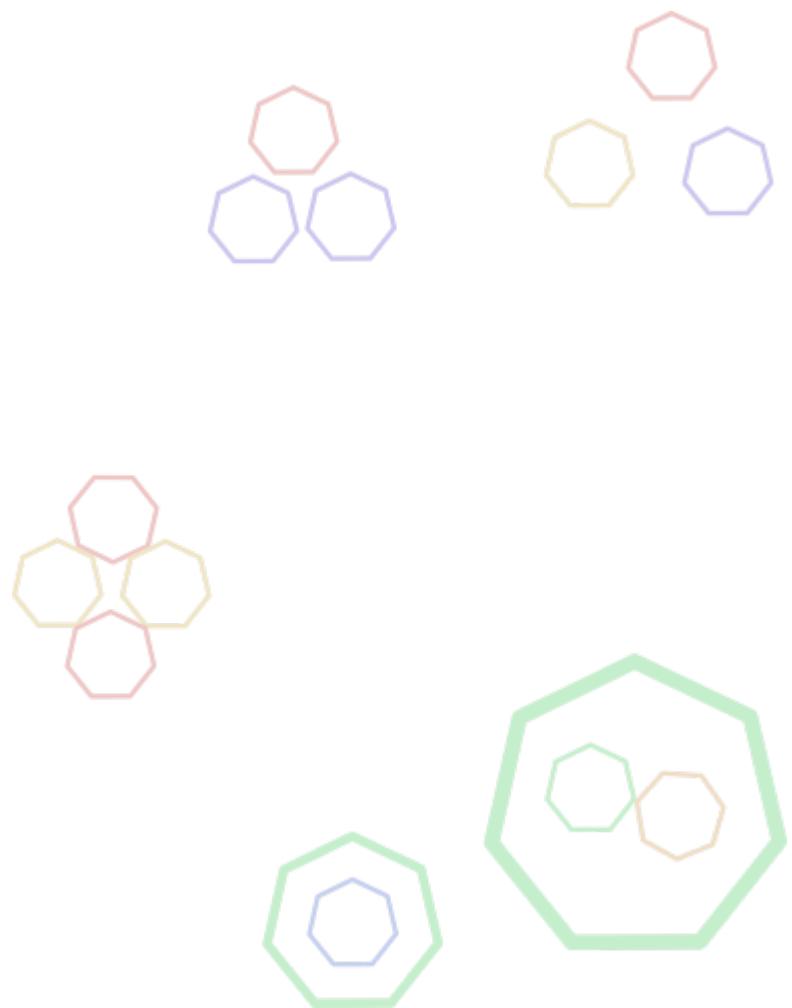
You can use 'y' to show YAML manifest in K9s

Object Labels

Labels can be used to filter which objects are displayed.

View all labels for objects:

```
$ kubectl -n kube-system get pod --show-labels
NAME                  READY   STATUS    RESTARTS   AGE   LABELS
aws-node-bjb48        1/1     Running   1          2h    controller-revision-
hash=673321540,k8s-app=aws-node,pod-template-generation=1
aws-node-mnk2m         1/1     Running   0          2h    controller-revision-
hash=673321540,k8s-app=aws-node,pod-template-generation=1
kube-dns-fcd468cb-wv9kp 3/3     Running   0          2h    eks.amazonaws.com/
component=kube-dns,k8s-app=kube-dns,pod-template-hash=97802476
kube-proxy-d2xj2       1/1     Running   0          2h    controller-revision-
hash=3853209624,k8s-app=kube-proxy,pod-template-generation=1
kube-proxy-xk9xn       1/1     Running   0          2h    controller-revision-
hash=3853209624,k8s-app=kube-proxy,pod-template-generation=1
```



Filtering Objects

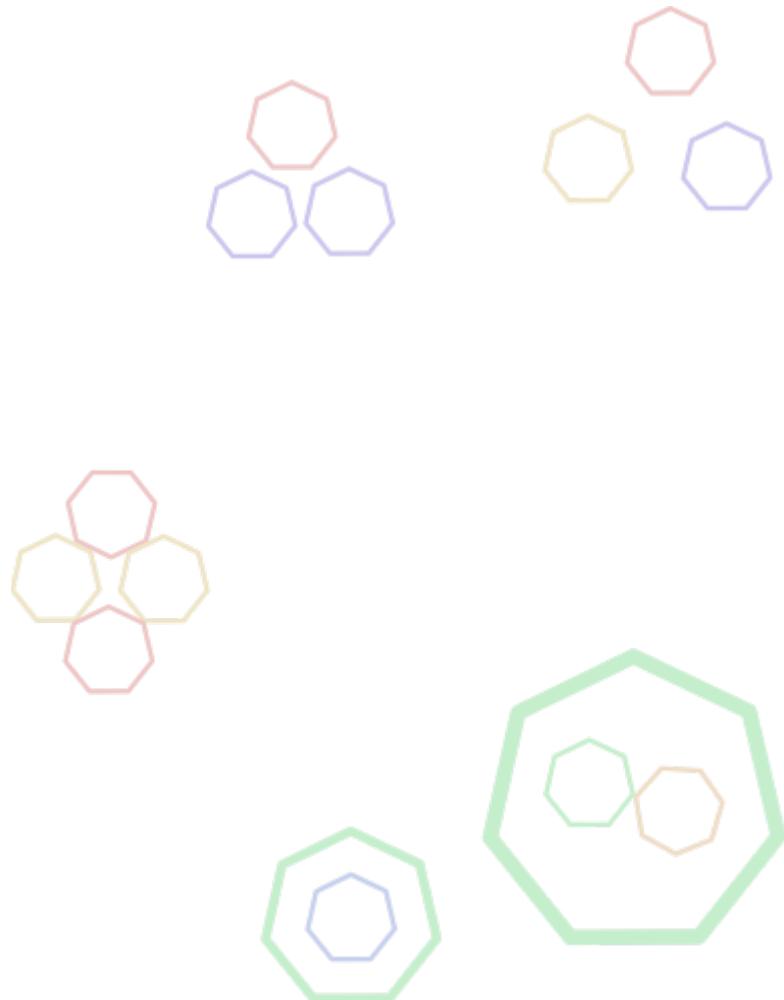
Filtering objects by label is useful because objects are linked to each other using labels:

```
$ kubectl -n kube-system get all -l k8s-app=kube-proxy
NAME           READY   STATUS    RESTARTS   AGE
pod/kube-proxy-d2xj2  1/1     Running   0          2h
pod/kube-proxy-xk9xn  1/1     Running   0          2h

NAME              DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR          AGE
daemonset.apps/kube-proxy  2         2         2        2            2           <none>      2h
```



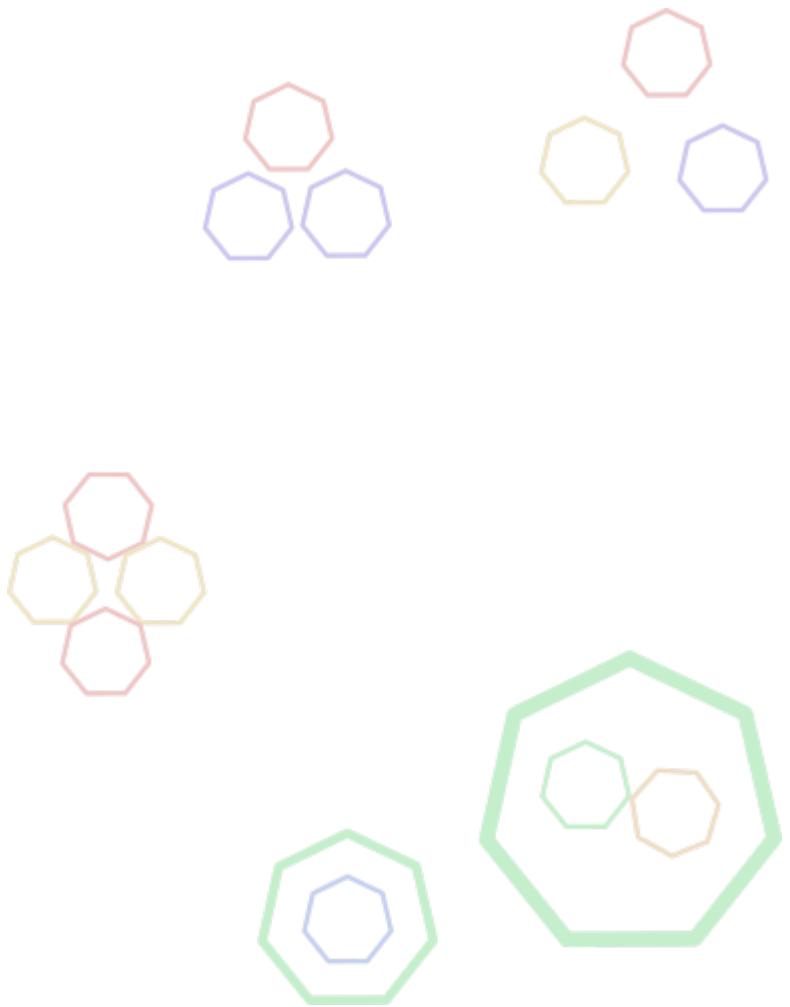
You can filter by labels in k9s with /app=web



Watching Objects

Use `--watch` to watch for object changes:

```
$ kubectl apply -f apache-pod.yaml && kubectl get pods --watch
NAME      READY   STATUS    RESTARTS   AGE
consumer  1/1     Running   0          44h
producer   1/1     Running   0          44h
apache     0/1     Pending   0          0s
apache     0/1     Pending   0          0s
apache     0/1     ContainerCreating   0          0s
apache     1/1     Running   0          8s
```



Describing Objects

`kubectl describe` displays details about specific objects (e.g. Event : start date, scaling info)

```
$ kubectl describe -n kube-system service kubernetes-dashboard
Name:           kubernetes-dashboard
Namespace:      kube-system
Labels:         k8s-app=kubernetes-dashboard
Selector:       k8s-app=kubernetes-dashboard
Type:          LoadBalancer
IP:            172.20.61.83
LoadBalancer Ingress: adcb6fca-31258.eu-west-1.elb.amazonaws.com
Port:          <unset> 443/TCP
TargetPort:    8443/TCP
NodePort:      <unset> 31737/TCP
Endpoints:     10.50.0.33:8443
Session Affinity: None
External Traffic Policy: Cluster
Events:
  Type  Reason          Age   From            Message
  ----  -----          ----  ----
  Normal  Type           16m   service-controller  ClusterIP ->
  LoadBalancer
  Normal  EnsuringLoadBalancer  16m   service-controller  Ensuring load balancer
  Normal  EnsuredLoadBalancer  15m   service-controller  Ensured load balancer
```



You can use 'd' to describe object in k9s

View Container Logs

```
$ kubectl logs -f <NAME>
```

- For multiple containers in a *Pod*, use `-c` to choose one container
- `--timestamps` adds date and time to log line

```
$ kubectl logs -n kube-system kubernetes-dashboard-7b9c7bc8c9-4bhxc
2018/11/05 13:00:59 Starting overwatch
2018/11/05 13:00:59 Using in-cluster config to connect to apiserver
2018/11/05 13:00:59 Using service account token for csrf signing
2018/11/05 13:01:00 Metric client health check failed: the server is currently unable
to handle the request (get services heapster). Retrying in 30 seconds.
2018/11/05 13:01:00 Auto-generating certificates
2018/11/05 13:01:00 Successfully created certificates
2018/11/05 13:01:00 Serving securely on HTTPS port: 8443
```



You can use 'l' to show logs in k9s on *Pod* and *Deployment*.

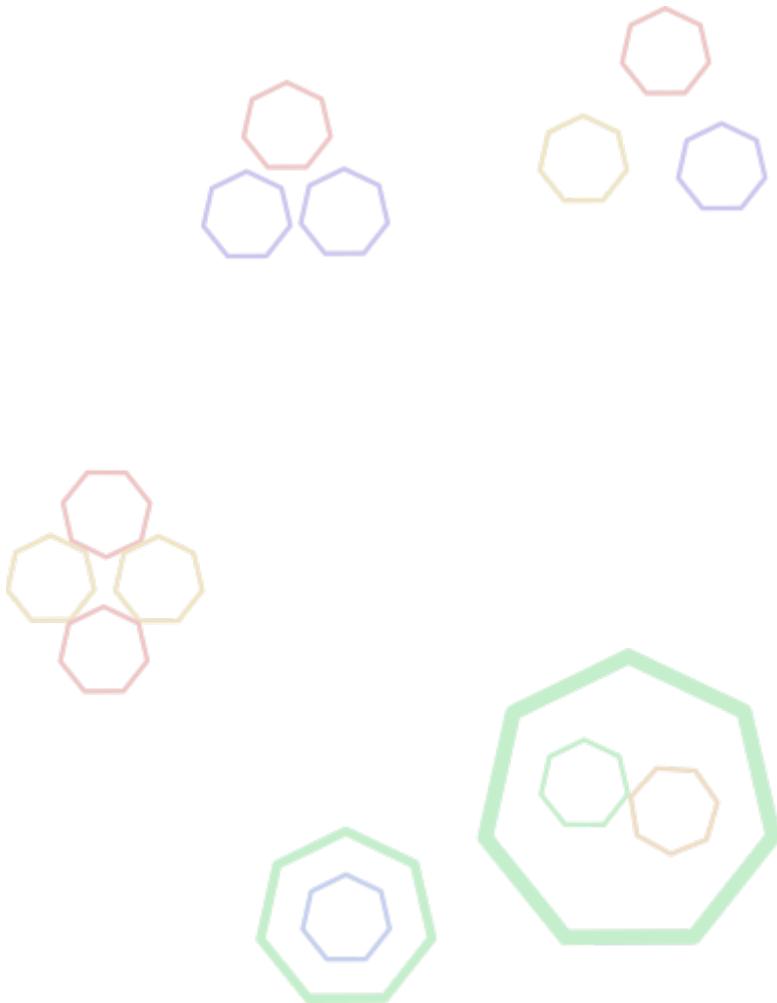
Intrusive Debugging Tools

Useful but not recommended in production (as containers should be immutable):

- `kubectl exec <NAME> -i -t bash`: open a new shell in container
- `kubectl attach <NAME>`: plug stdin and stdout to the entrypoint
- `kubectl cp <SRC> <DST>`: copy files from/to containers
- `kubectl port-forward <NAME> <LOCAL PORT>:<REMOTE PORT>`



K9s has shortcuts to create *PortForwards* on *Services* and *Pods*



Accessing Internal Services

- When Services are not published, `kubectl` can create a proxy to access the API
- All Services can be accessed through the API:

```
$ kubectl proxy &
Starting to serve on 127.0.0.1:8001
$ xdg-open http://127.0.0.1:8001/
```

- Services are available at:

`http://localhost:8001/api/v1/namespaces/<namespace>/services/[https:]<service:name>:[<port>]/proxy/`

- The proxy can be published on other ports/interfaces, e.g.:

```
$ kubectl proxy --address=0.0.0.0 -p 8080
```

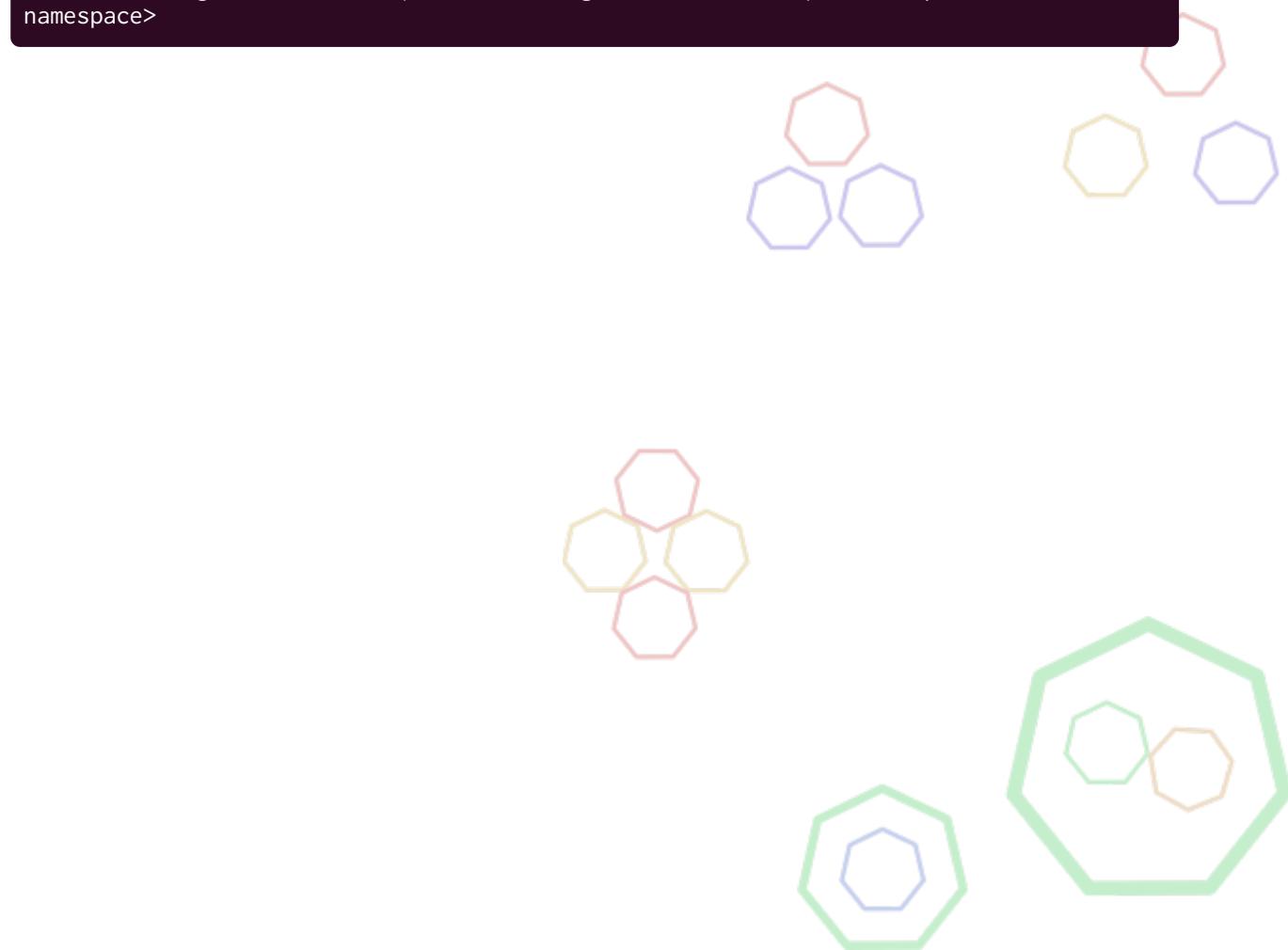


Namespaces

Provide Object isolation:

- Multiple projects/users on the same hardware cluster
- Objects of the same type must have different names in one namespace
- Deleting a namespace deletes all objects in it
- `kubectl get` has a `--all-namespaces` option
- All commands have a `-n/--namespace` option
- Setting default namespace:

```
kubectl config set-context $(kubectl config current-context) --namespace=<default namespace>
```



Lab 19.3: Create a Namespace and set it as default



Objective

- Avoid conflicts with other users of the cluster

Steps

- Create a new *Namespace* with your login as name
- Configure `kubectl` to use it as default
- Test default *Namespace* with:

```
$ kubectl config view | grep namespace:
```

Lab 19.4: Load Pod and Service and debug it



Objective

- Create objects in Kubernetes
- Modify *Pod* after deployment

Steps

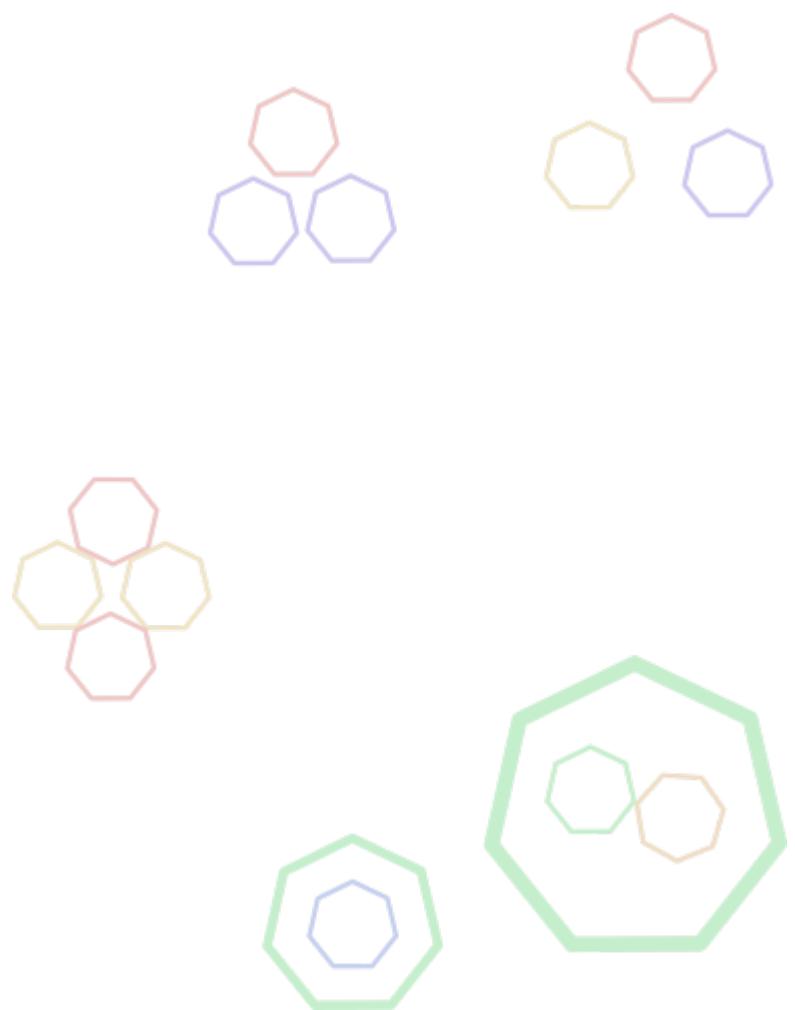
- Create a *Pod* and a *Service* and access the PHP server
- Retrieve the DNS entry to access the Service, wait for DNS record to be propagated
- Fix the syntax error in the `index.php` script
- Delete *Pod* and *Service* to remove load balancer

Questions

- Can we use `kubectl exec` to modify the PHP script?
 - yes
 - no
- Can we use `kubectl cp` to modify the PHP script?
 - yes
 - no
- Can we use vim to modify the PHP script?
 - yes
 - no

Checkpoint: Command Line Interface

- Object values can be extracted using
 - JsonPath expressions
 - jq
 - Go template queries
- Objects can be filtered on
 - their type
 - their namespace
 - their labels
 - their annotations
 - a regex matching their name
- What advantages are there to using a declarative workflow for Objects?
 - idempotent management
 - object versioning
 - objects can be updated
 - objects can be destroyed
 - rolling upgrades



KUBERNETES API

Lesson 20: API

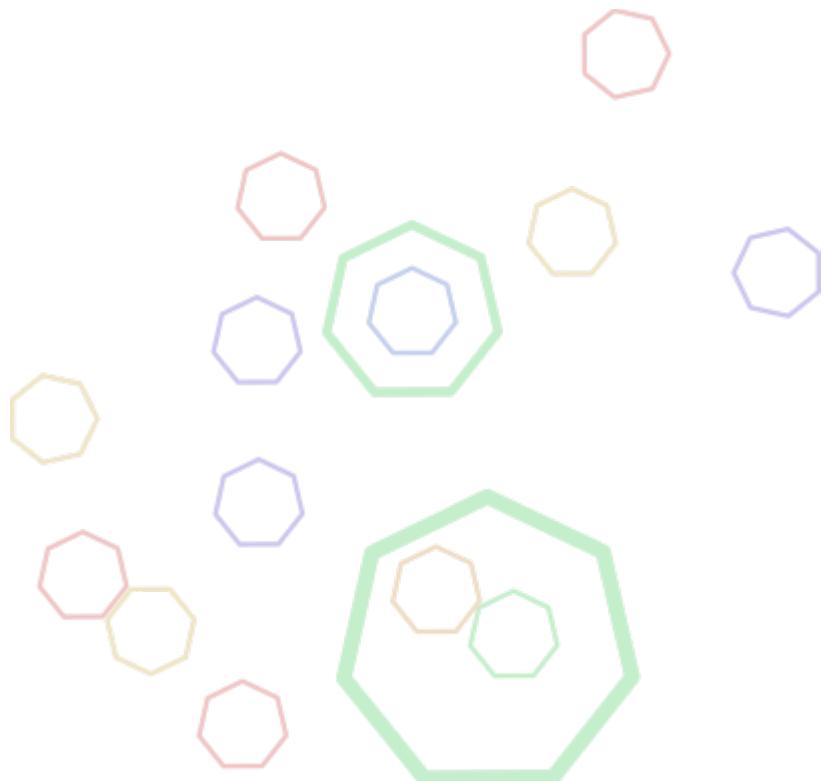
Objectives

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

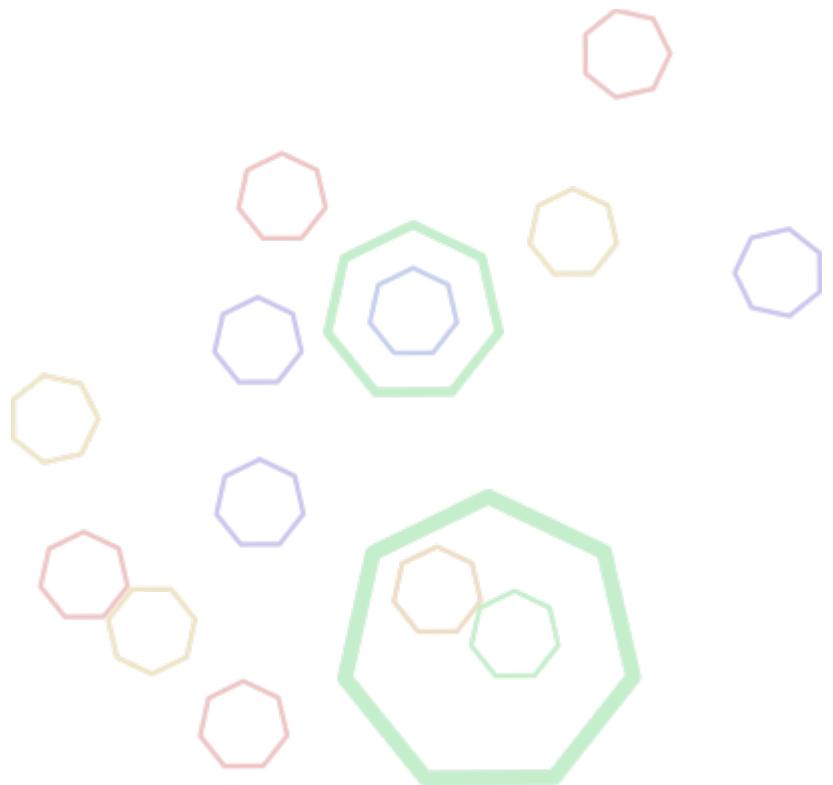
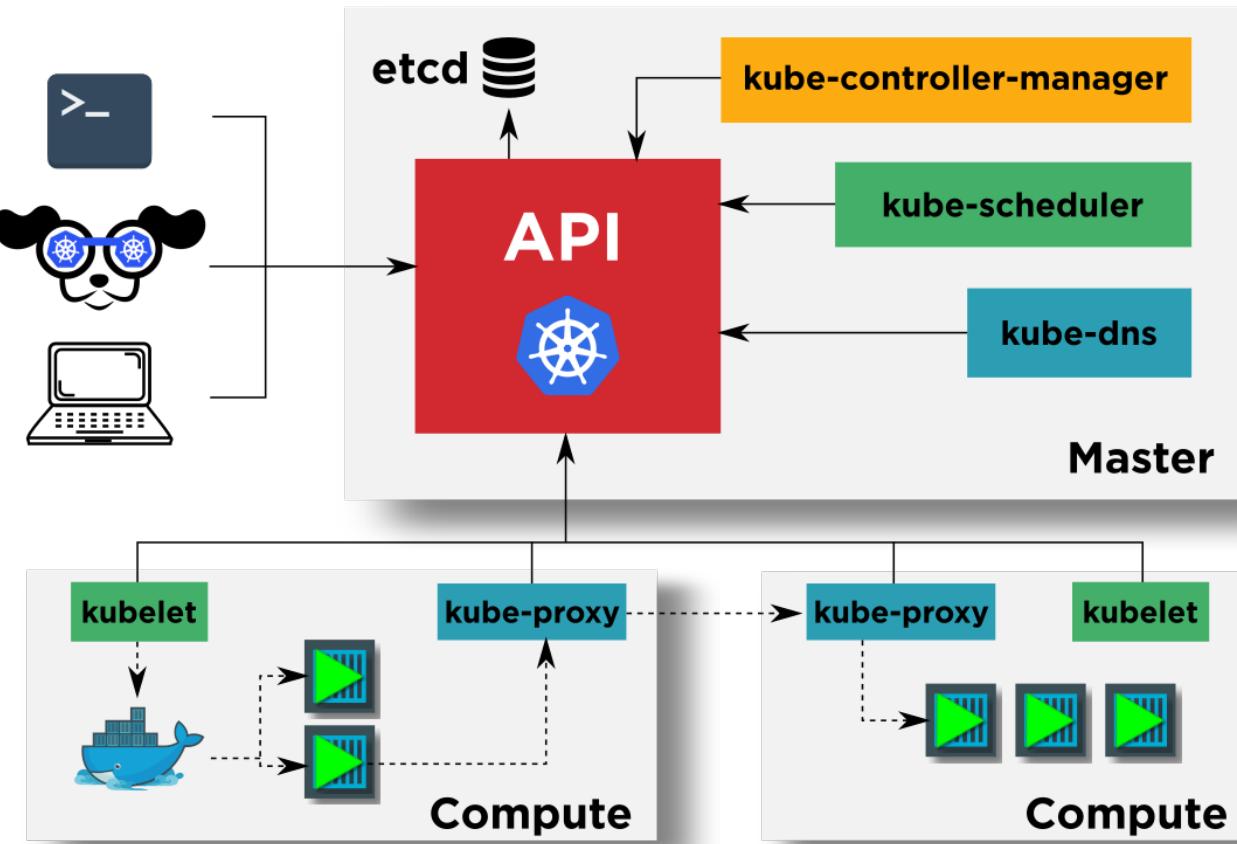
- understand how the Kubernetes API is organized
- query the API
- understand labels and annotations on Kubernetes Objects

API Basics

- **Unique** entrypoint to interact with the orchestrator
 - kubectl
 - kubeadm
 - dashboard
 - client libraries
- HTTP REST API documented on <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/>



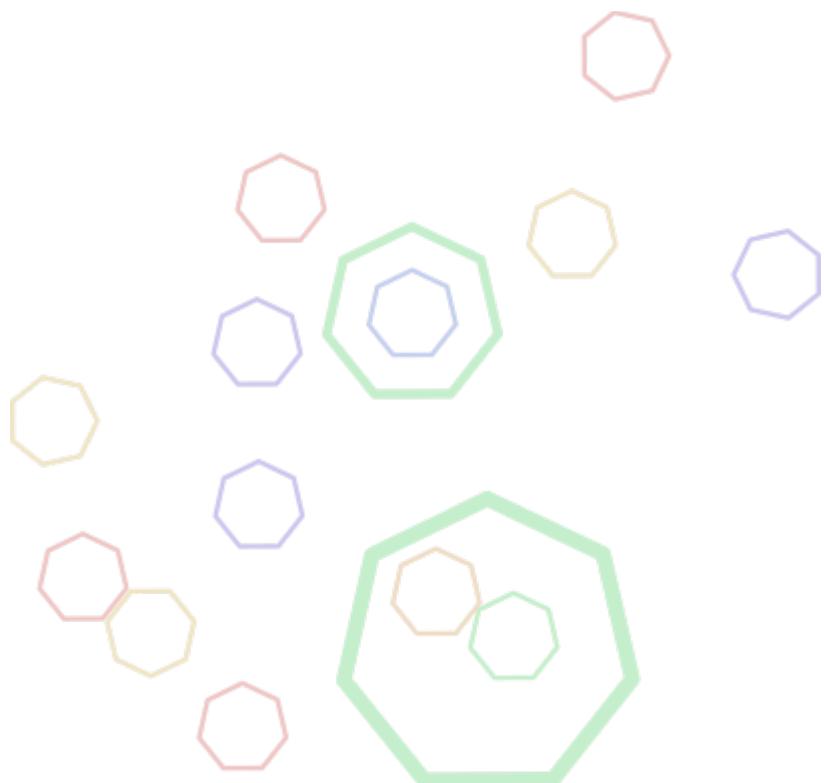
API Basics



Everything is an Object

In Kubernetes, everything is described as API objects:

- Workload: *Pod, CronJob, Deployment*
- Network: *Service, Endpoints, Ingress*
- Config and storage: *PersistentVolume, Secret, ConfigMap*
- Cluster: *Node, Role, Namespace*

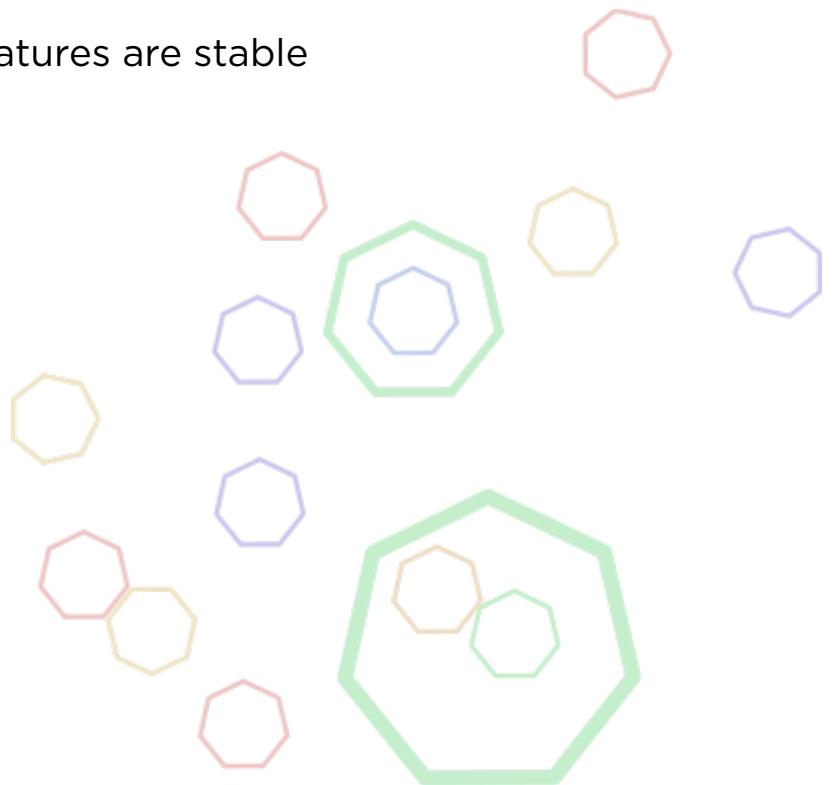


API Versioning

The API is divided by version to manage object lifecycle.

Different API versions indicate different levels of stability and support.

- *alpha*: e.g. /api/v1alpha1
 - introduces new features
 - features can be removed or changed without notice
- *beta*: e.g. /api/v2beta3
 - consolidates existing alpha features
 - features can change, with migration instructions
- *stable*: e.g. /api/v1
 - final feature release
 - API will not change and features are stable



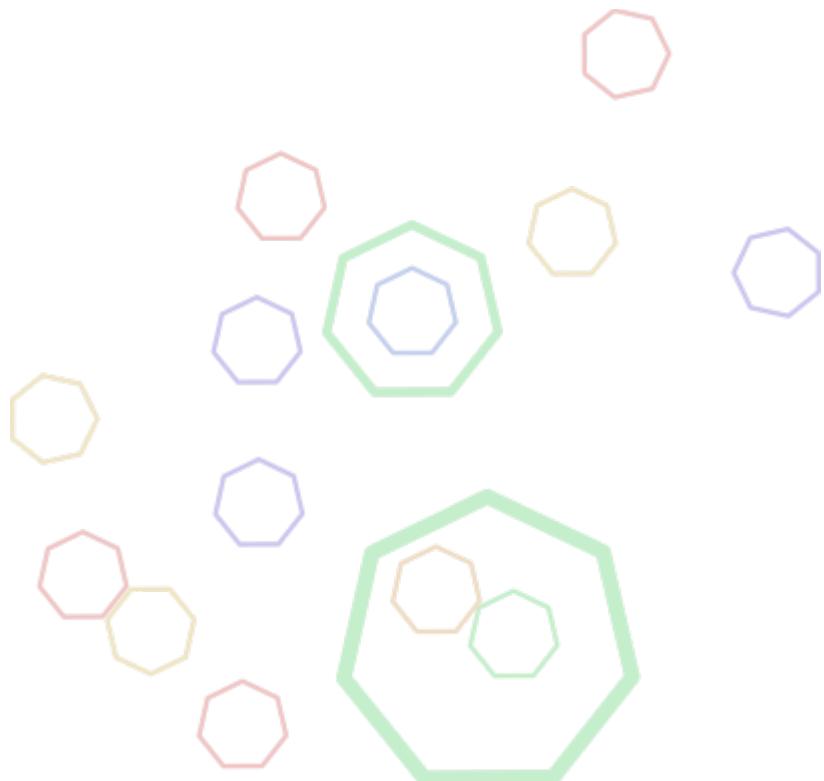
API Groups

Core Group

- Located at `/api/<VERSION>`
- Contains core Kubernetes objects

Named Groups

- Located at `/apis/<GROUP>/<VERSION>`
- Contains Kubernetes extensions
- Examples:
 - `apps` provides *Deployment*
 - `batch` provides *Job, CronJob*
 - `storage.k8s.io` provides *StorageClass*



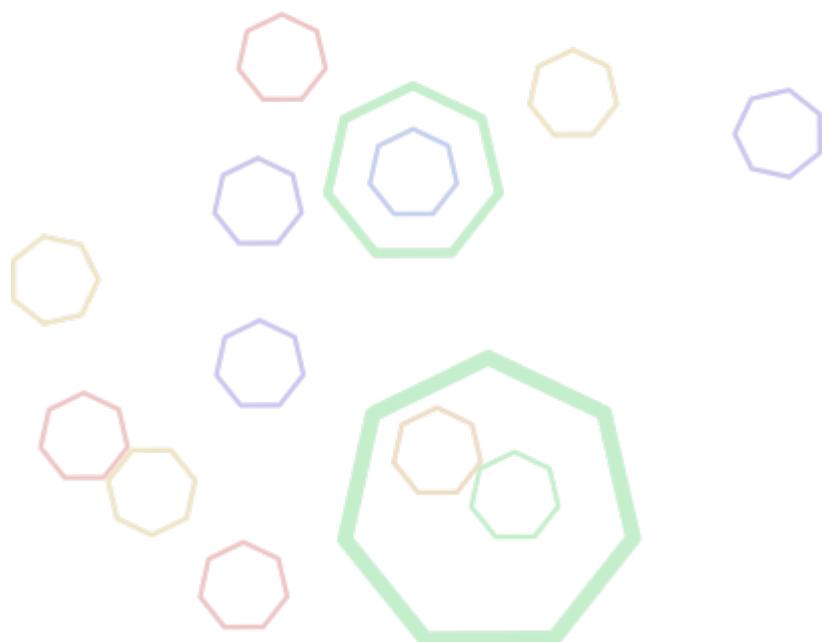
API Deprecations

A [migration guide](#) is provided when APIs become deprecated

Examples:

- Deployment: `extensions/v1beta1` → `apps/v1beta1` → `apps/v1beta2`
→ `apps/v1`
- PodSecurityPolicy: `extensions/v1beta1` → `policy/v1beta1` → `retired`
- Ingress: `extensions/v1beta1` → `networking.k8s.io/v1beta1` → `networking.k8s.io/v1`

```
@@ -1,4 +1,4 @@
-apiVersion: extensions/v1beta1
+apiVersion: networking.k8s.io/v1
 kind: Ingress
 metadata:
   name: apache
@@ -9,5 +9,8 @@
   paths:
   - path: /
     backend:
-       serviceName: apache
-       servicePort: 80
+       service:
+         name: test
+         port:
+           number: 80
```



API versions in YAML manifests

Specified as:

```
apiVersion: [<GROUP>/]<VERSION>
```

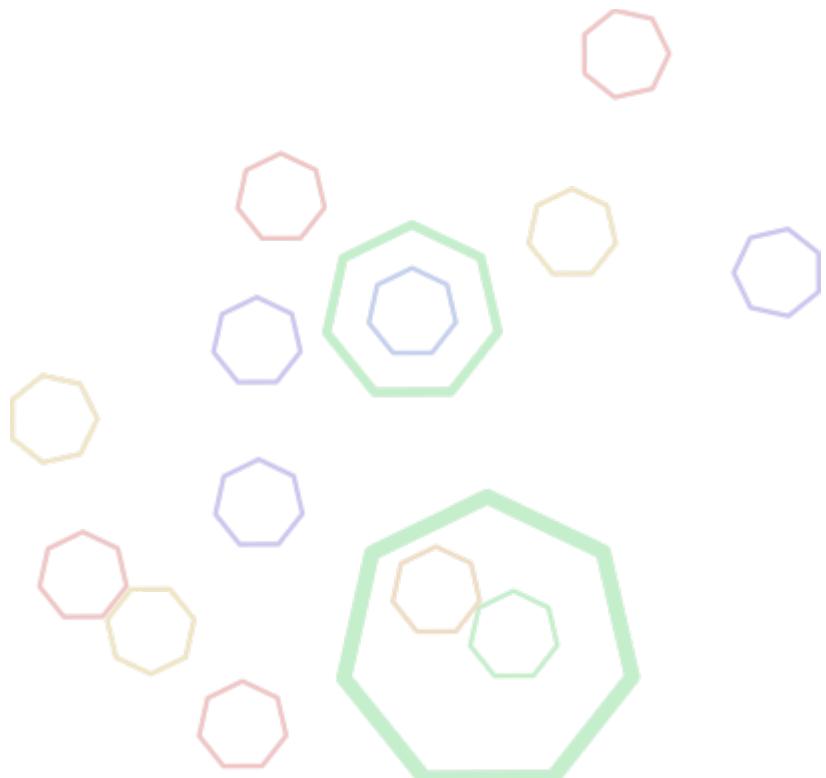
Examples:

- **Pod**: *Group: core, Version: v1, Kind: Pod*

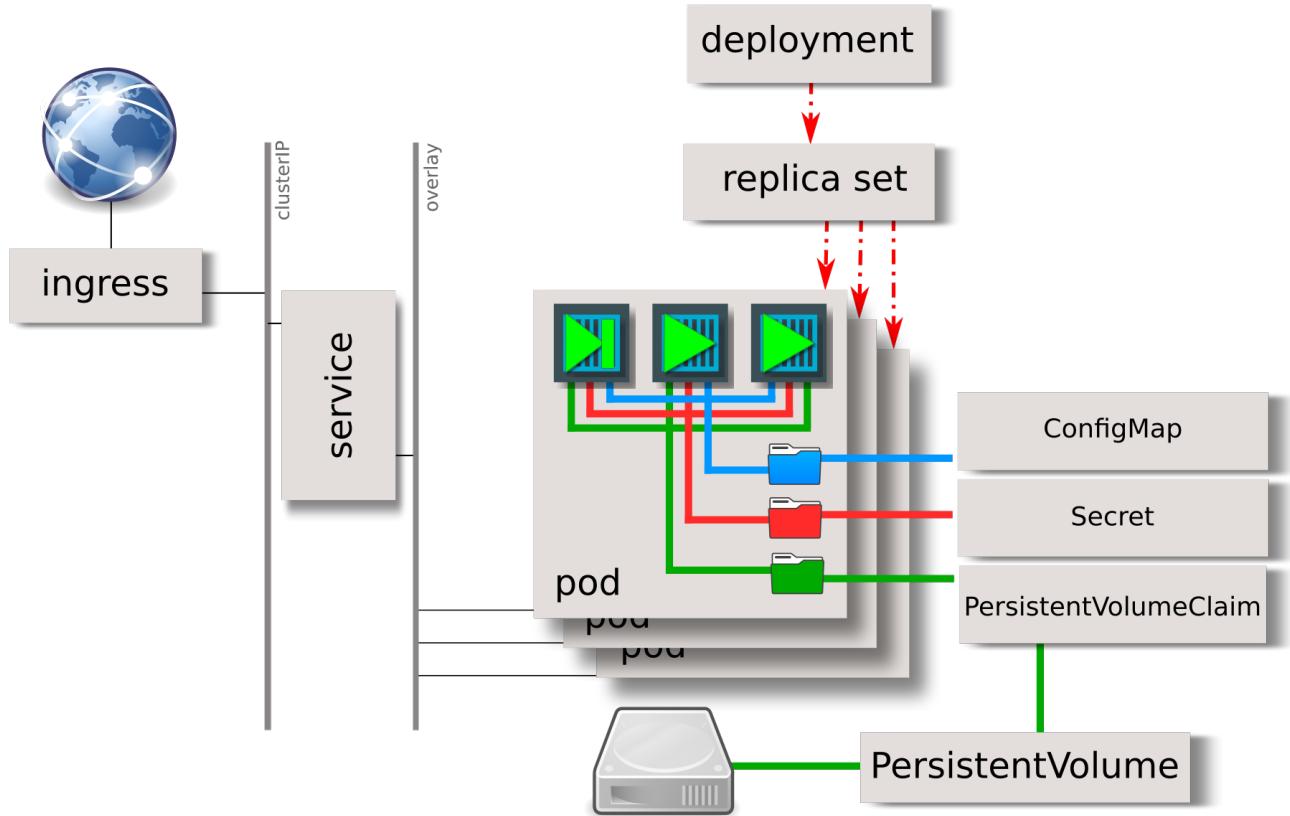
```
kind: Pod  
apiVersion: v1
```

- **Deployment**: *Group: apps, Version: v1, Kind: Deployment*

```
kind: Deployment  
apiVersion: apps/v1
```



API Objects Overview

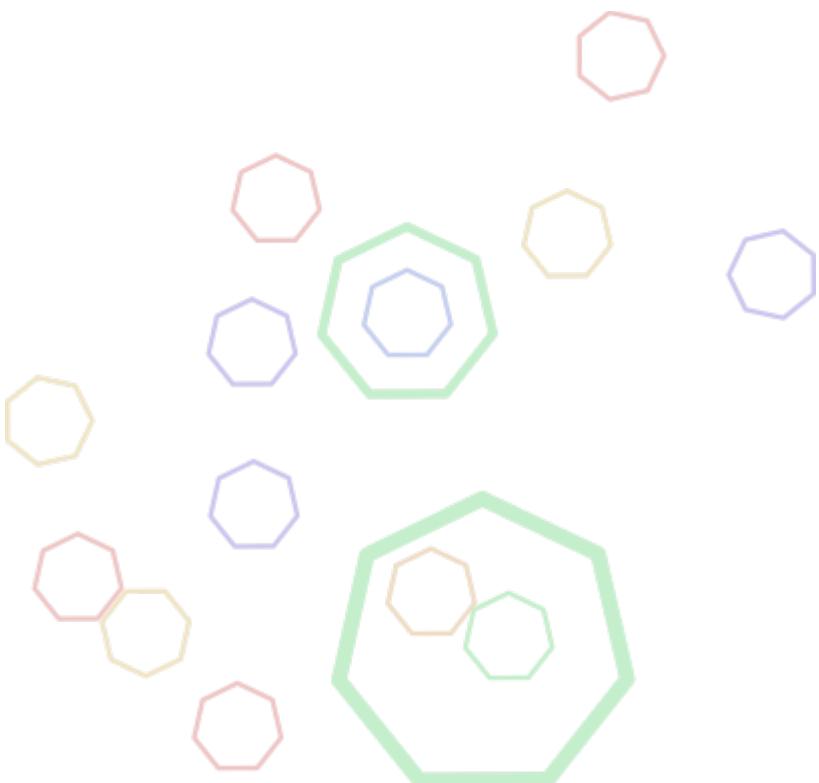


Containers and Volumes are not API objects (they are *Pod* fields)

YAML Manifests

- Objects are described in JSON or YAML format.
- Example of a *Pod* definition:

```
kind: Pod
apiVersion: v1
metadata:
  name: apache
spec:
  containers:
    - name: apache
      image: httpd:2.4
```



Lab 20.1: Configure your editor



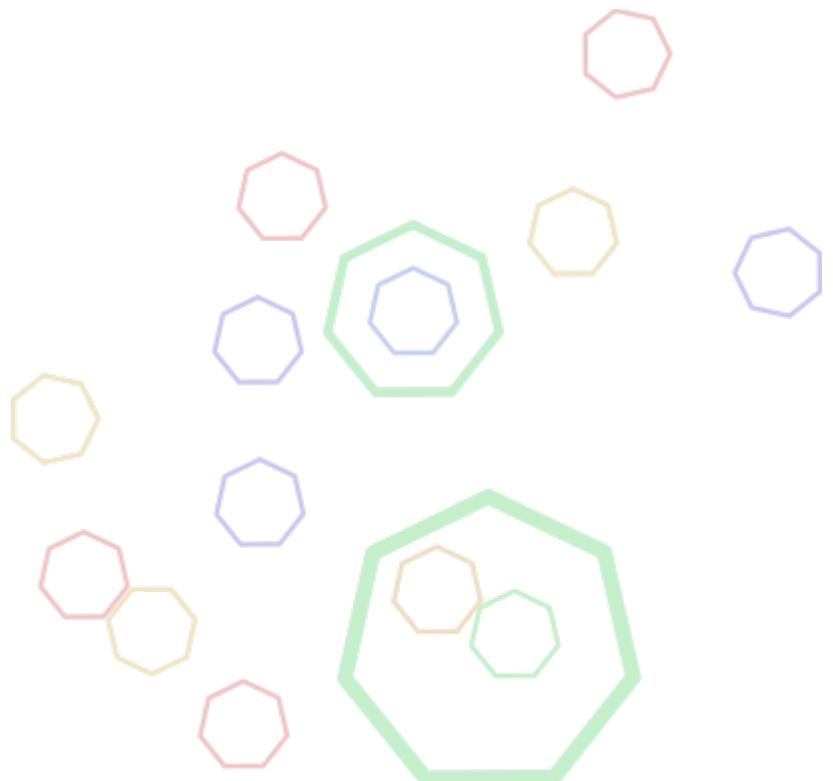
Objective

- Configure your editor for YAML edition

Steps

- Open Visual Studio Code
- Browse to the Extensions tab (`Ctrl+Shift+X`)
- Search for the official Kubernetes Extension and install it
- Open the Command Palette and look for the "Kubernetes: Watch" action
- Inspect the newly opened Kubernetes tab

Questions



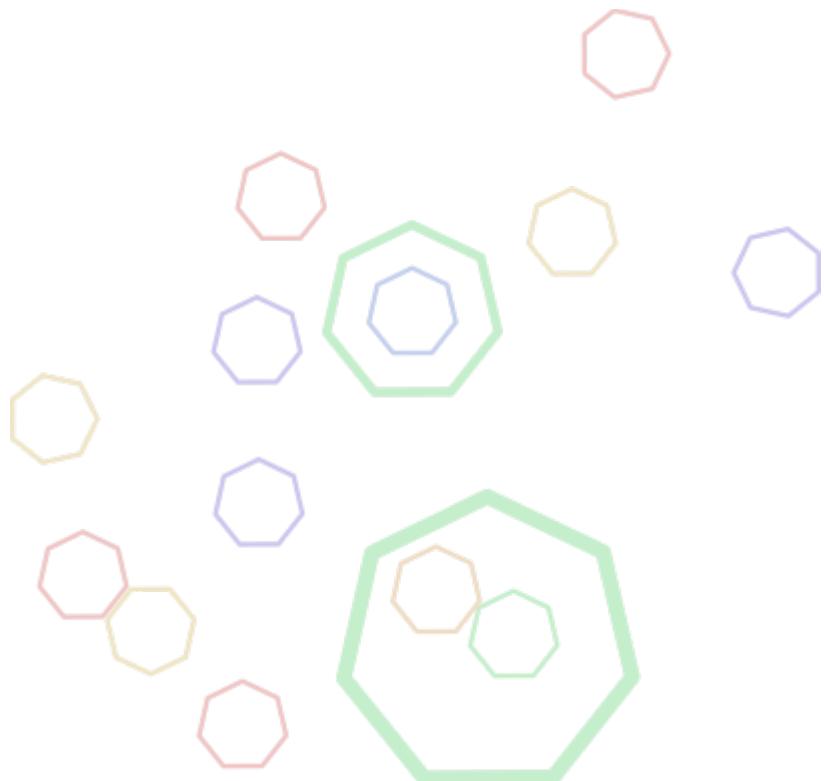
Deployments

Deployments are high-level objects (Controllers) that control base objects:

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
spec:
  replicas: 1
  selector:
    app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
        - name: apache
          image: httpd:2.4
```



Object names must be unique for one object type



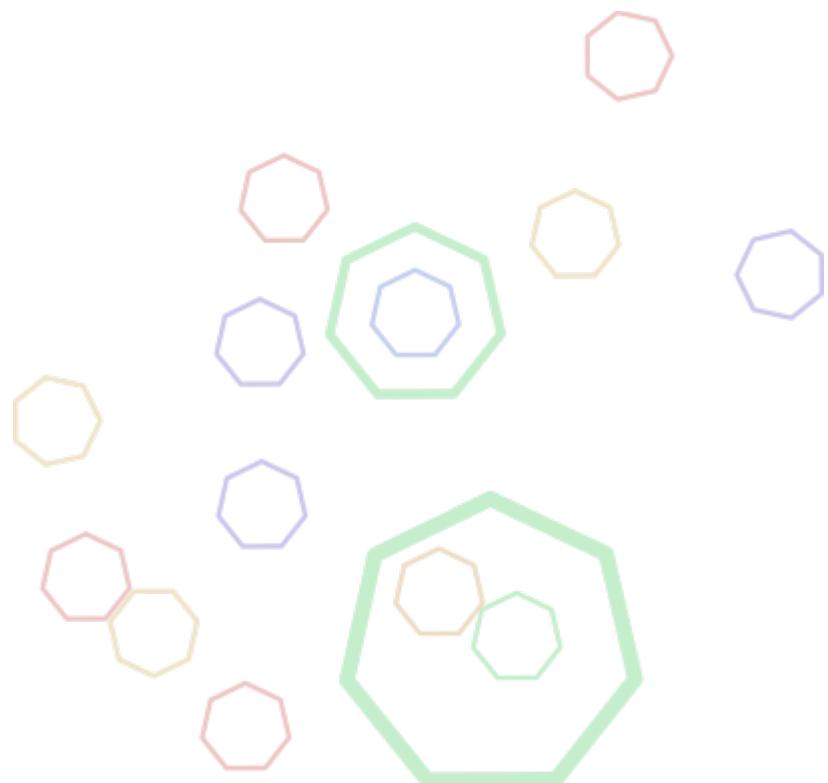
Objects Labels

- Key/value pairs at `metadata.labels`
- Meaningful and readable
- Many objects can share the same label



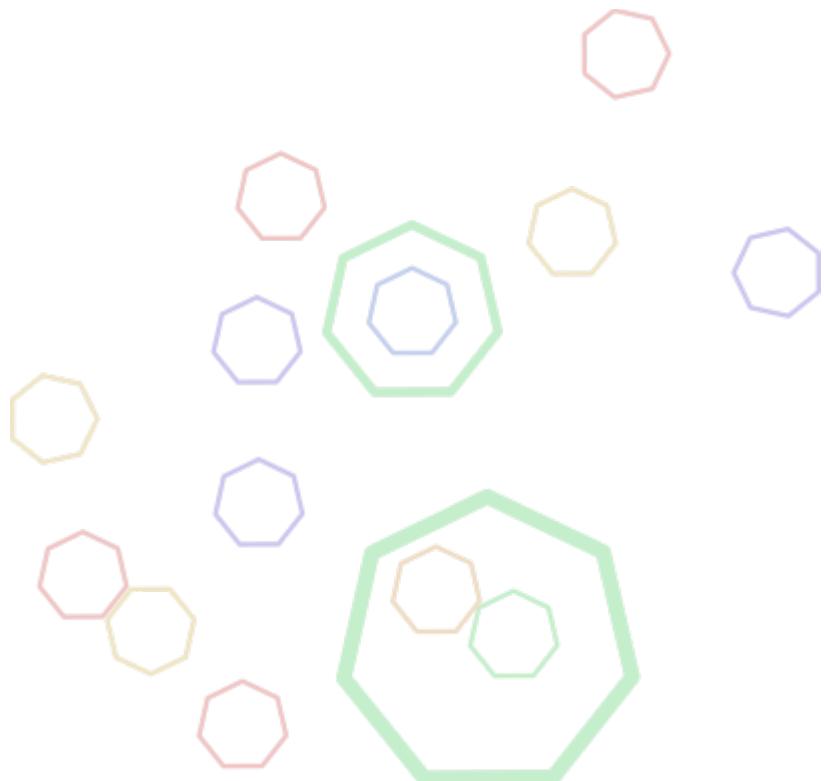
Labels are **the only way** to group objects

- *ReplicaSets* and *Services* use labels to find their *Pods*
- Metadata on objects: release, version, environment
- Node labels for affinity
- `kubectl` can filter objects by label



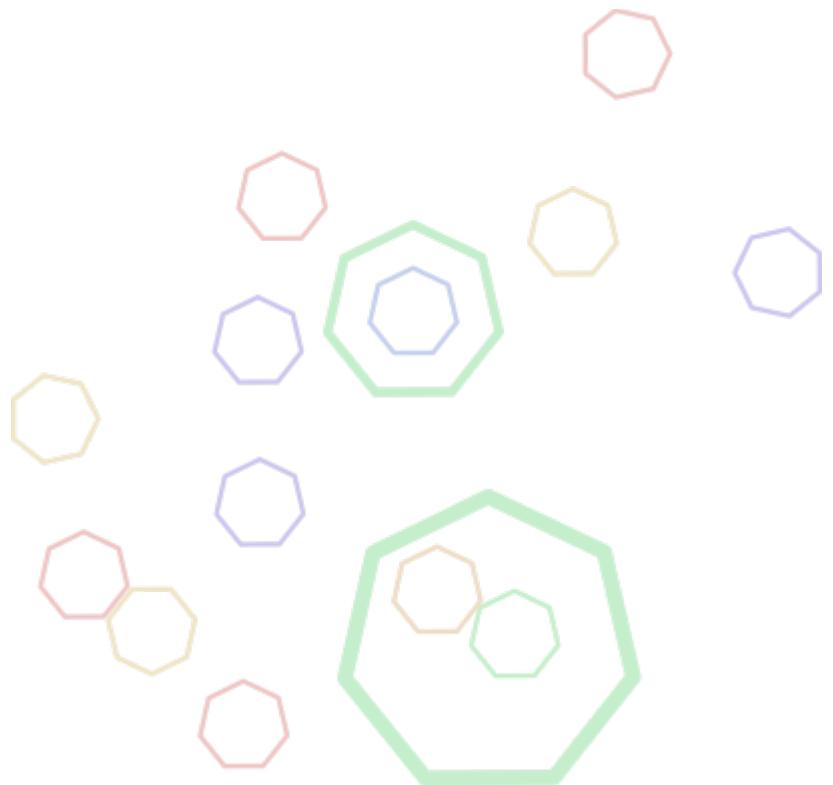
Objects Labels Specification

- 63 character max
- First and last character must be alphanumeric
- Can contains `-`, `_`, `.`
- Can have a prefix (max 253 characters)
 - Prefix must be a DNS entry



Annotations

- Some features are activated with annotations (e.g. SSL routes)
- complex/structured data



API Documentation

`kubectl` provides a detailed API documentation with the `explain` command:

```
$ kubectl explain <type>[.<field>]
```

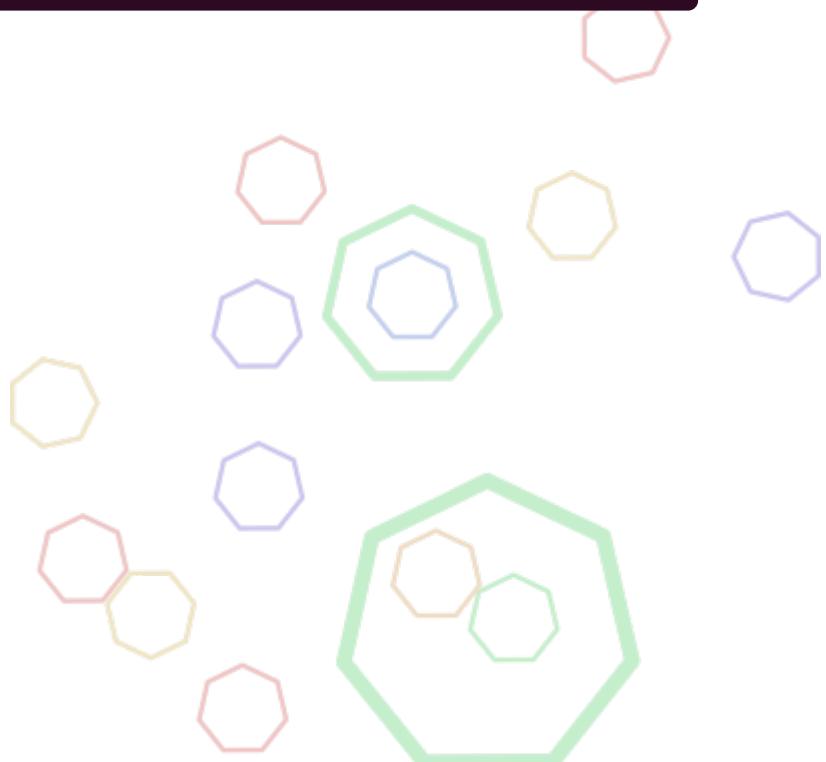
Example:

```
$ kubectl explain pod.spec.containers
KIND:     Pod
VERSION:  v1

RESOURCE: containers <[]Object>

DESCRIPTION:
  List of containers belonging to the pod. Containers cannot currently be
  added or removed. There must be at least one container in a Pod. Cannot be
  updated.

FIELDS:
  args <[]string>
    Arguments to the entrypoint. The docker image's CMD is used if this is not
    provided. Variable references $(VAR_NAME) are expanded using the
```



Exercise 20.2: Find API Documentation

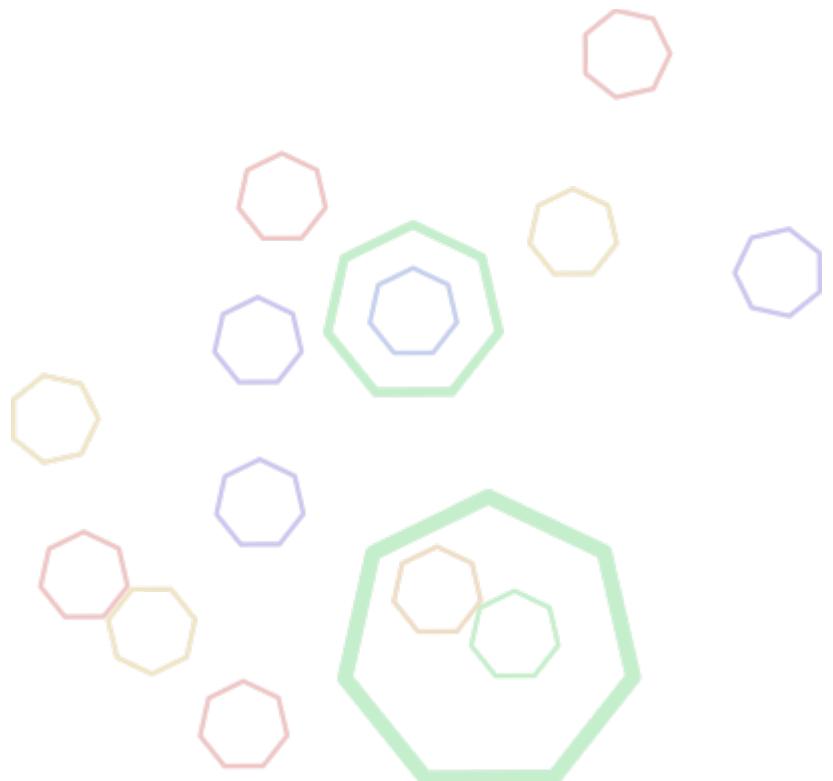


Objective

- Browse API doc to find documentation about specific fields

Steps

- Using `kubectl explain`, find valid values for the followings fields:
 - `spec → containers → imagePullPolicy` of *Pod*
 - `spec → containers → tty` of *Pod*
 - `spec → volumes` of *Pod*
- In Visual Studio Code:
 - open an object from the Kubernetes tab
 - hover on YAML fields to see their definitions

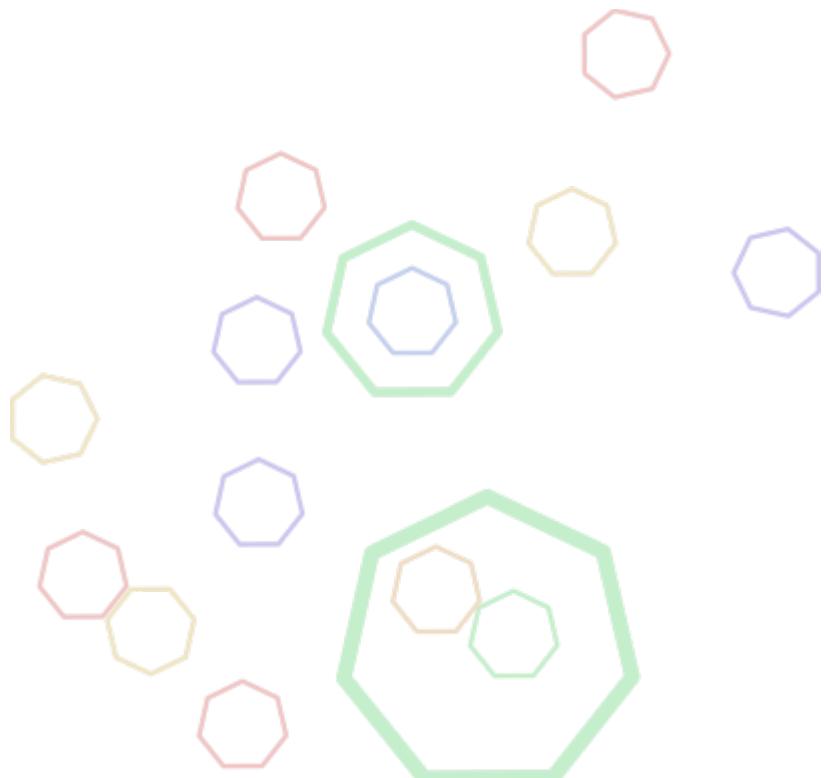


Custom Resources

The API is extensible, using *CustomResourceDefinitions* (CRDs)

- Defines new Custom Resource (CRs) types
- Can be defined as YAML
- The new CRs can be instantiated just like native objects

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: acme-crt
spec:
  secretName: acme-crt-secret
  dnsNames:
  - foo.example.com
  - bar.example.com
  issuerRef:
    name: letsencrypt-prod
    kind: Issuer
    group: cert-manager.io
```



API Authentication

Authentication based on SSL certificates

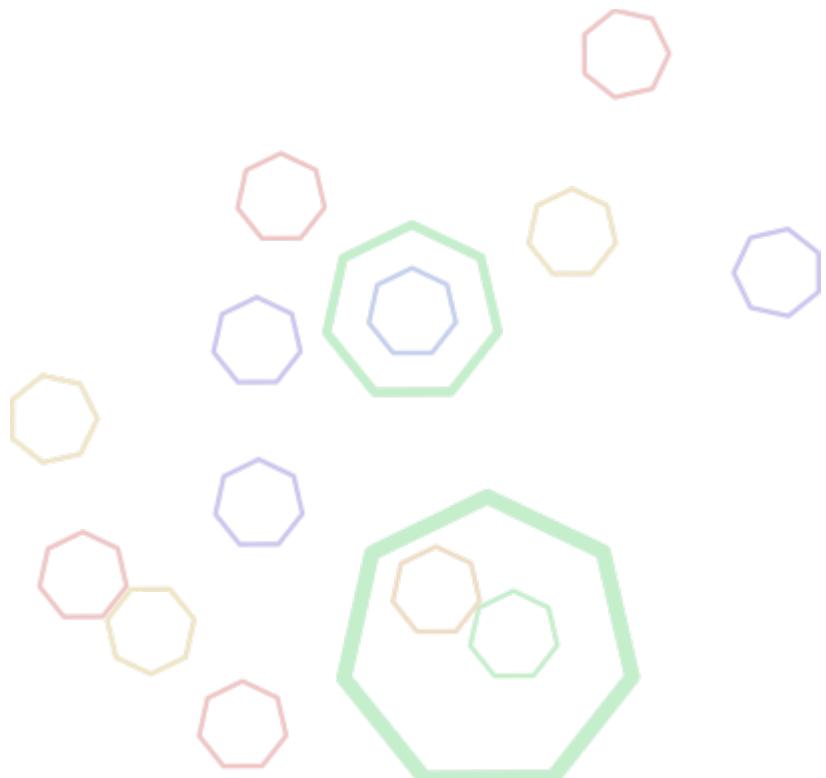
- Requests as:
 - normal user (managed by external system)
 - service account (managed by Kubernetes, bound to namespace)
 - anonymous user
- SSL certificates signed by Kubernetes CA
- CN determines user name



Service Accounts

- Internal user accounts
- Managed by Kubernetes (as API objects)
- Namespaced

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: showoff
  namespace: course
```



Roles

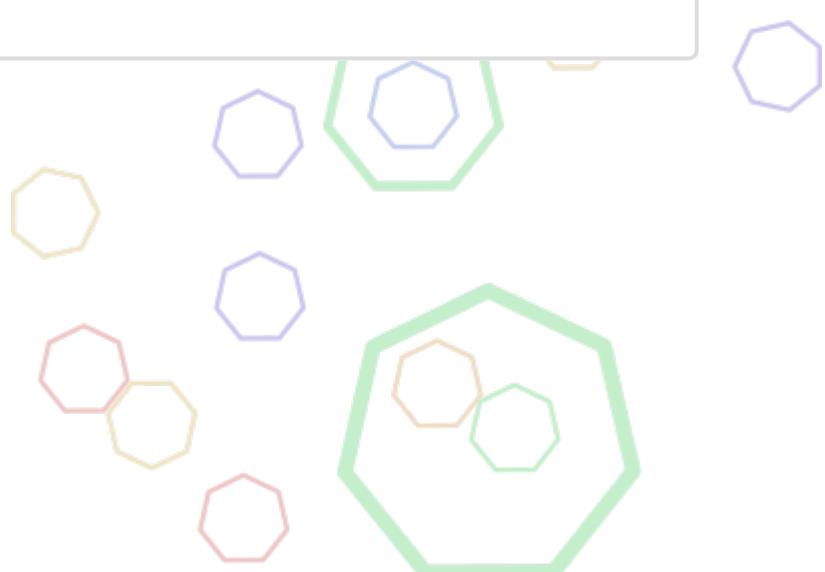
Two types:

- *Role* (namespaced)
- *ClusterRole* (global)

Specify authorizations, with rules linking:

- resources (API object types)
- verbs (actions)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: cert-manager-webhook:dynamic-serving
  namespace: cert-manager
rules:
- apiGroups:
  - ""
    resourceNames:
    - cert-manager-webhook-ca
    resources:
    - secrets
    verbs:
    - get
    - list
    - watch
    - update
- apiGroups:
  - ""
    resources:
    - secrets
    verbs:
    - create
```



Role Bindings

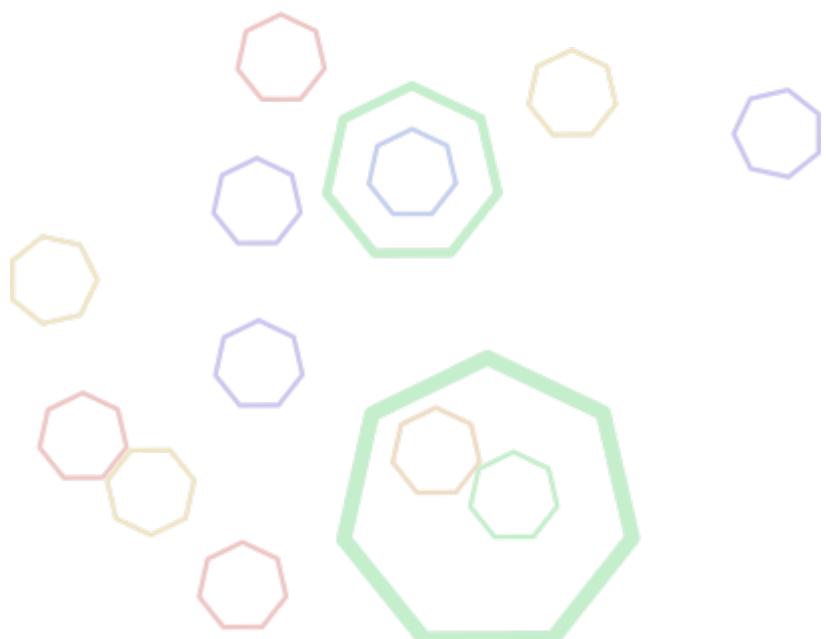
Two types:

- *RoleBinding* (links *Roles*)
- *ClusterRoleBinding* (links *ClusterRoles*)

Associates a *Role* with a subject, among:

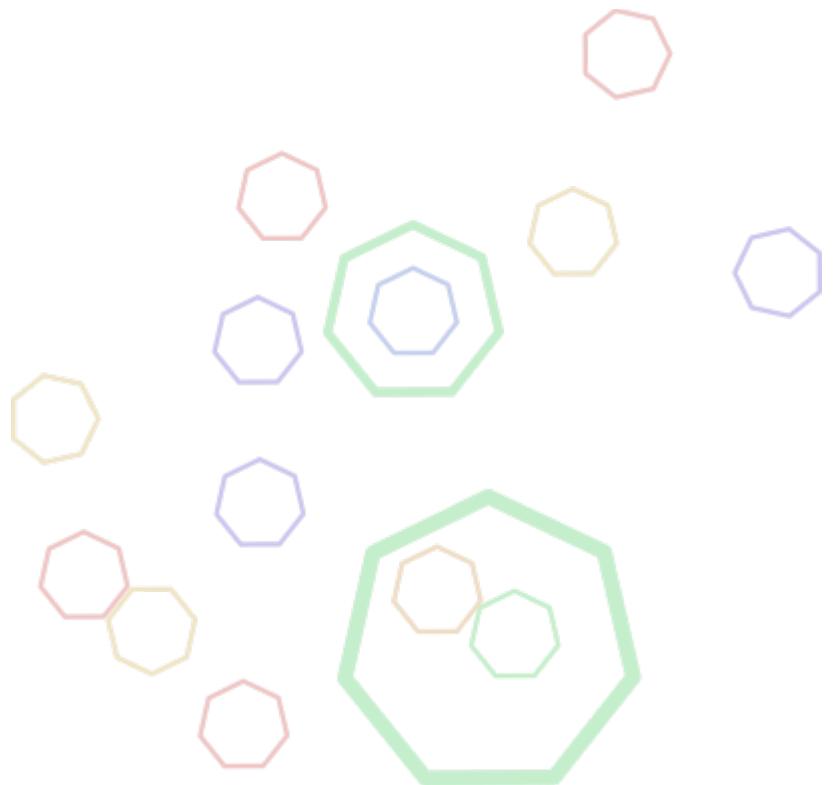
- *ServiceAccounts*
- *Users*
- *Groups*

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    name: cert-manager-webhook:dynamic-serving
  namespace: cert-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: cert-manager-webhook:dynamic-serving
subjects:
- kind: ServiceAccount
  name: cert-manager-webhook
  namespace: cert-manager
```



Checkpoint: API

- Can the API version of k8s objects change between k8s release?
 - yes
 - no
- Can a feature be part of multiple API groups?
 - yes
 - no
- Can the same label be defined on two objects?
 - yes
 - no



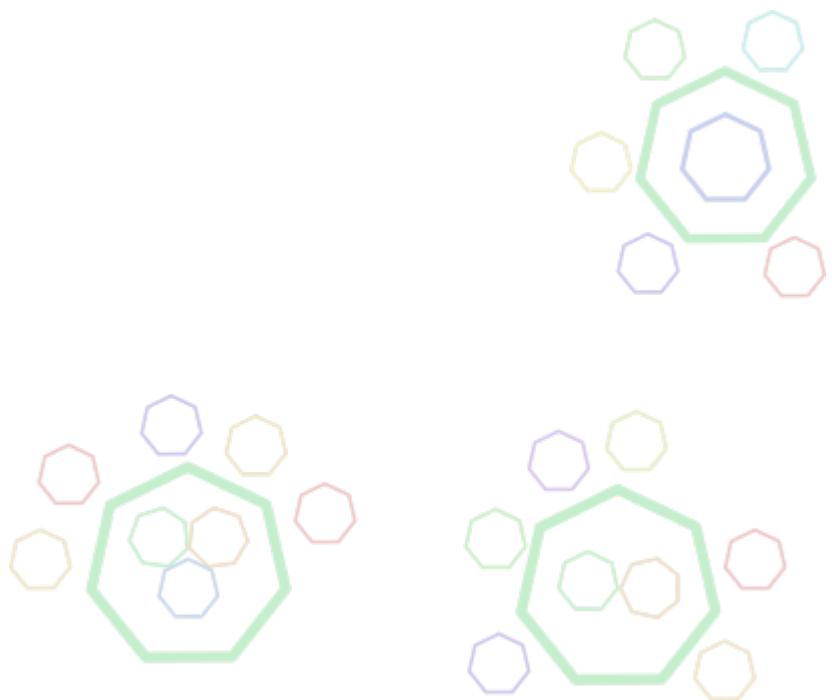
PODS

Lesson 21: Pods

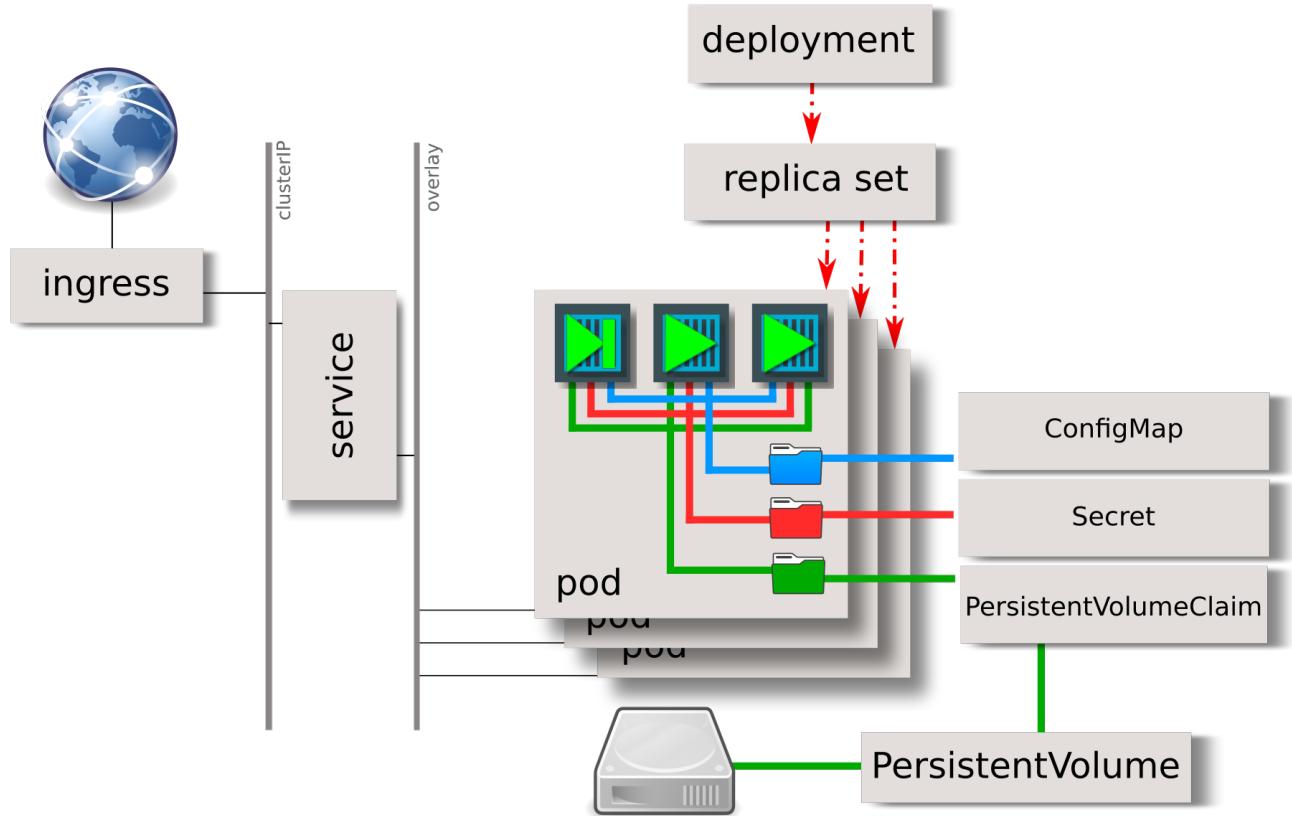
Objectives

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

- Understand what is a Pod
- Create pod and manage Pod
- Understand a Pod lifecycle



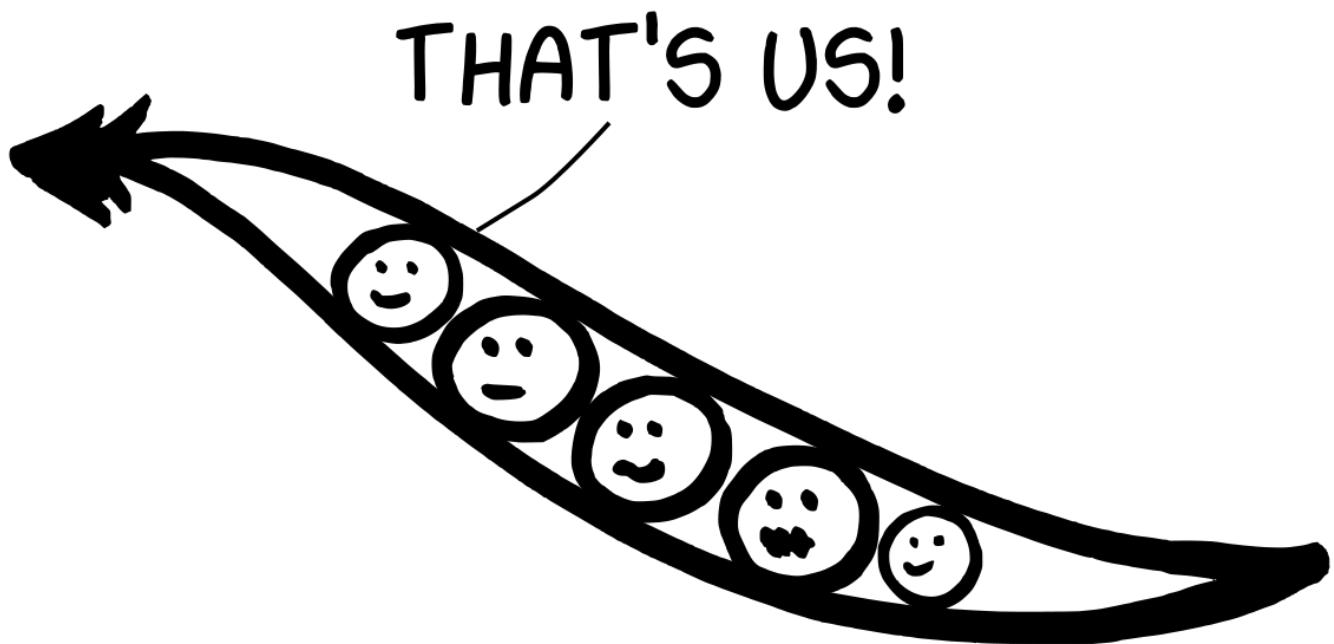
API Objects Overview



What is a Pod?

Pods are colocated containers (same mounts, same network)

- One or more containers
- Provides only one micro service
- May include sidekicks
- Optionally ordered list of Init Containers



Notes:

- In most cases, a Pod only contains one Container

Pod Syntax

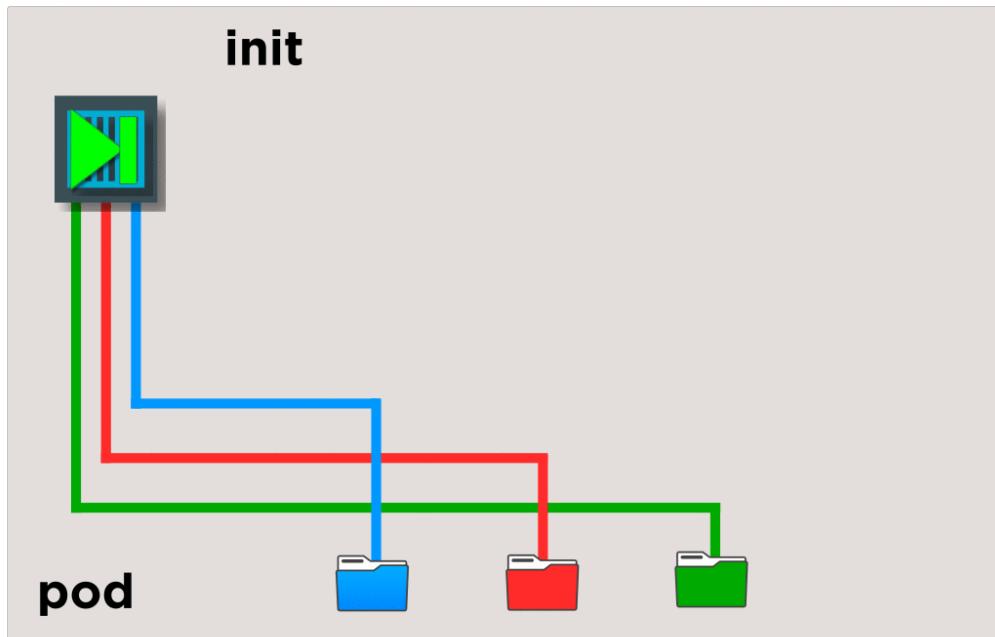
As YAML manifest:

```
kind: Pod
apiVersion: v1
metadata:
  name: producer
spec:
  containers:
    - name: producer
      image: producer
    - name: conf
      image: producer-conf
```



Pod Lifecycle

- Kubernetes chooses one node to attach the Pod to
- Init containers are started sequentially (success required)
- Finally, the "main" containers are started in parallel
- Readiness & Liveness Check defines the status of the Pod



Configuring Containers

You can configure each container using:

- Environment variables
- Command/args (equivalent of entrypoint/command in Docker)

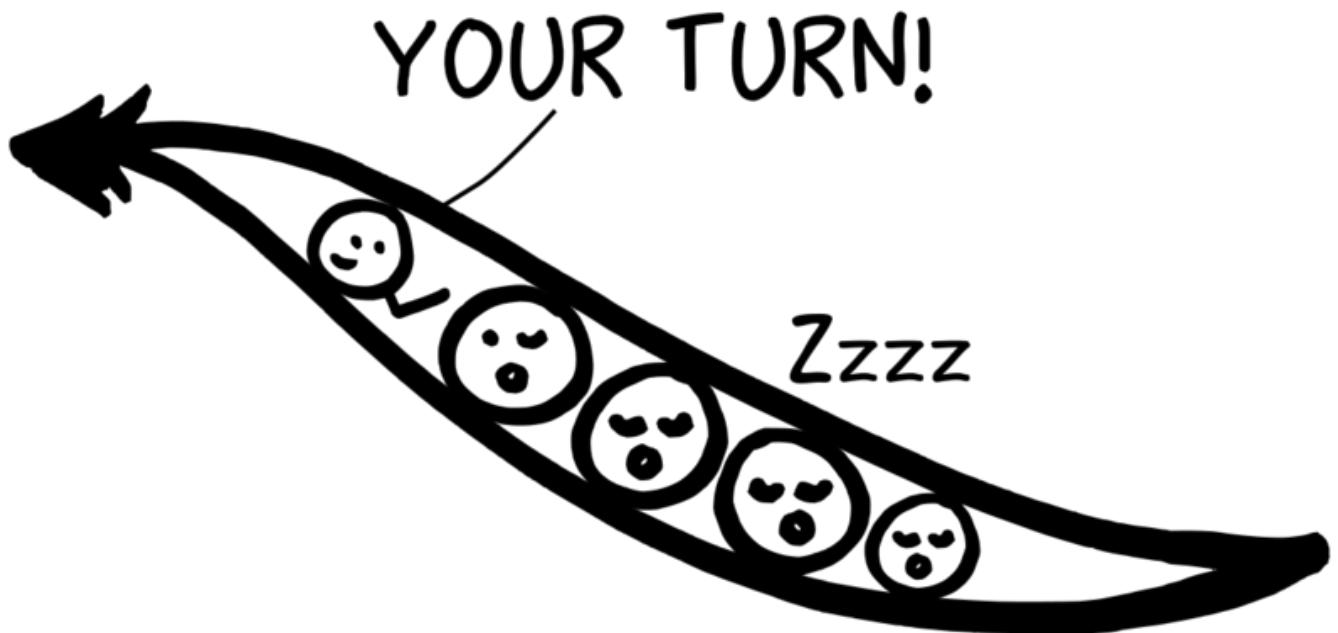
```
kind: Pod
apiVersion: v1
metadata:
  name: frontend
spec:
  containers:
    - name: frontend
      image: frontend:latest
      # container entrypoint
      command:
        - apache2ctl
      # container command
      args:
        - -DFOREGROUND
      env:
        - name: BACKEND_HOST
          value: backend
        - name: BACKEND_PORT
          value: '8080'
```



Environment variables must be string values!

Init Containers

- Use upstream image as-is
- Move bootstrap procedure into Init container
- Possibly make running container read-only



Why use Init Containers?

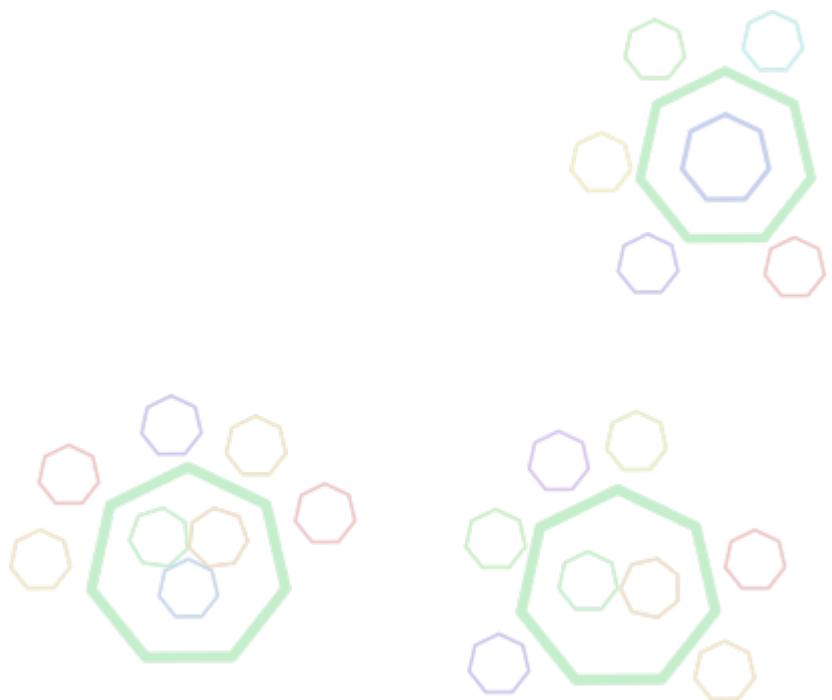
- Configure app (using e.g. `confd`)
- Bootstrap volumes for first launch
- Initialize code/data (with e.g. `git clone/pull`)
- Wait for other services' readiness (using e.g. `wait-for-it`)



Init Container Syntax

As YAML manifest:

```
---
kind: Pod
apiVersion: v1
metadata:
  name: producer
spec:
  initContainers:
    - name: init-myservice
      image: busybox
      command: ['sh', '-c', 'until nslookup myservice; do sleep 2; done;']
    - name: init-mydb
      image: busybox
      command: ['sh', '-c', 'until nslookup mydb; do sleep 2; done;']
  containers:
    - name: producer
      image: producer
    - name: conf
      image: producer-conf
```



Init Container: Environment Variables

- Populate config file in container image from env vars:

```
domain_name = ${DOMAIN}  
email_from = contact@${DOMAIN}
```

- Set Entrypoint/command:

```
sed -i "s/\${DOMAIN}/$DOMAIN/" /etc/app/app.conf
```

or using `envsubst`:

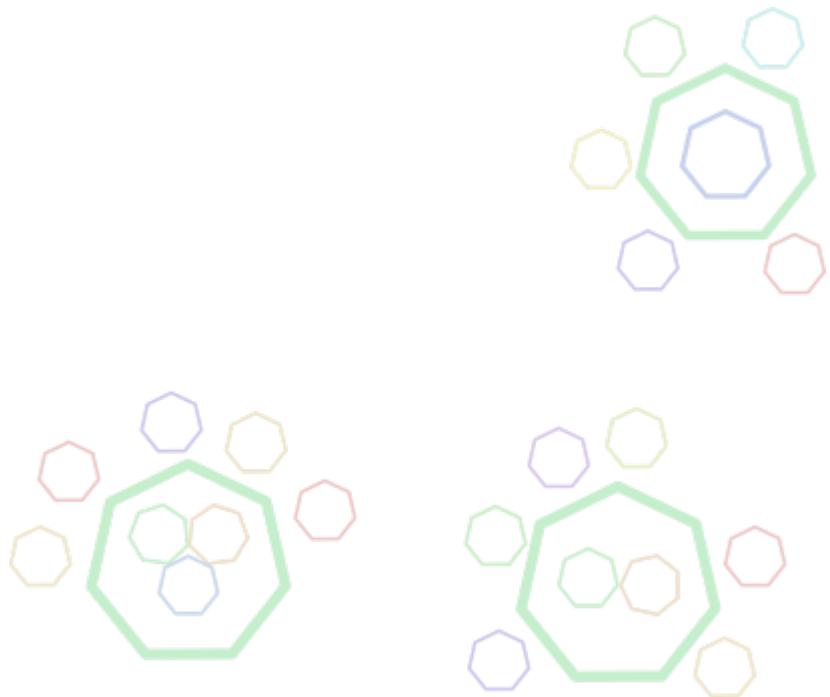
```
envsubst < app.conf > /etc/app/app.conf
```



Init Container: Shared Volume

Application and Init-domain containers need to use volumes to share configuration:

```
kind: Pod
apiVersion: v1
metadata:
  name: producer
spec:
  initContainers:
  - name: init-domain
    image: busybox
    command:
      - 'sh'
      - '-c'
      - 'sed -i "s/\${DOMAIN}/${DOMAIN}/" /mnt/app/app.conf'
    env:
      - name: DOMAIN
        value: example.com
    volumeMounts:
      - name: conf
        mountPath: /mnt/app
  containers:
  - name: app
    image: company/app:latest
    volumeMounts:
      - name: conf
        mountPath: /etc/app
  volumes:
  - name: conf
    emptyDir: {}
```



Init Container: Wait for Service

Example of entrypoint with <https://github.com/vishnubob/wait-for-it>:

```
#!/bin/bash
while true; do
    echo > /dev/tcp/database/5432
    if [ $? -eq 0 ]; then
        break;
    fi
    sleep 1
done
```



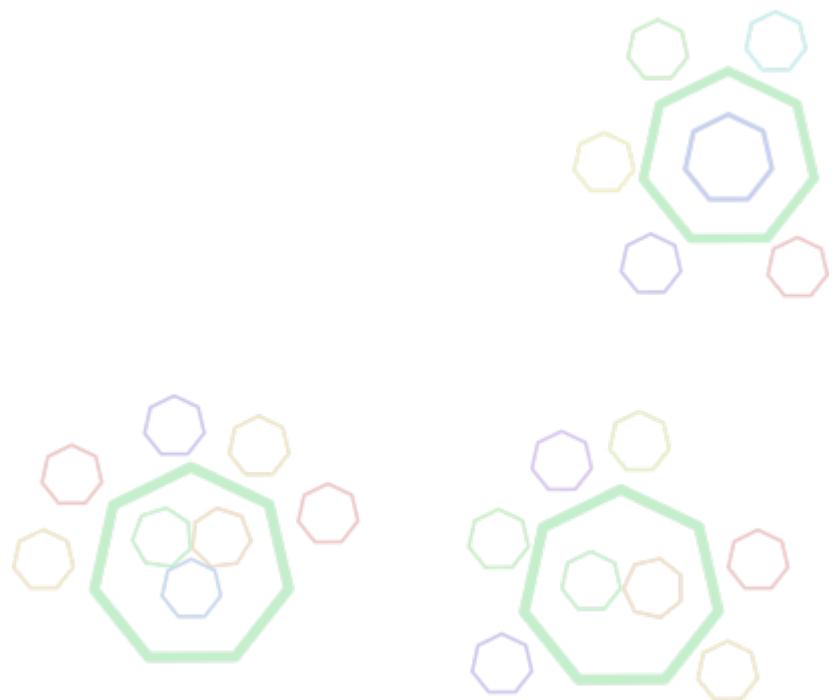
Common Pod Layout (1/3)

Sidecar container

Additional container can be added to a *Pod* to extend fonctionnality of the main container:

- periodic synchronisation
- log shipping

Sidecar containers are helpers to better integrate main container.



Common Pod Layout (2/3)

Ambassador container

An Ambassador container act as a proxy in front of the main container:

- Add authentication
- Implements retry, cache
- Log requests
- proxy to external service

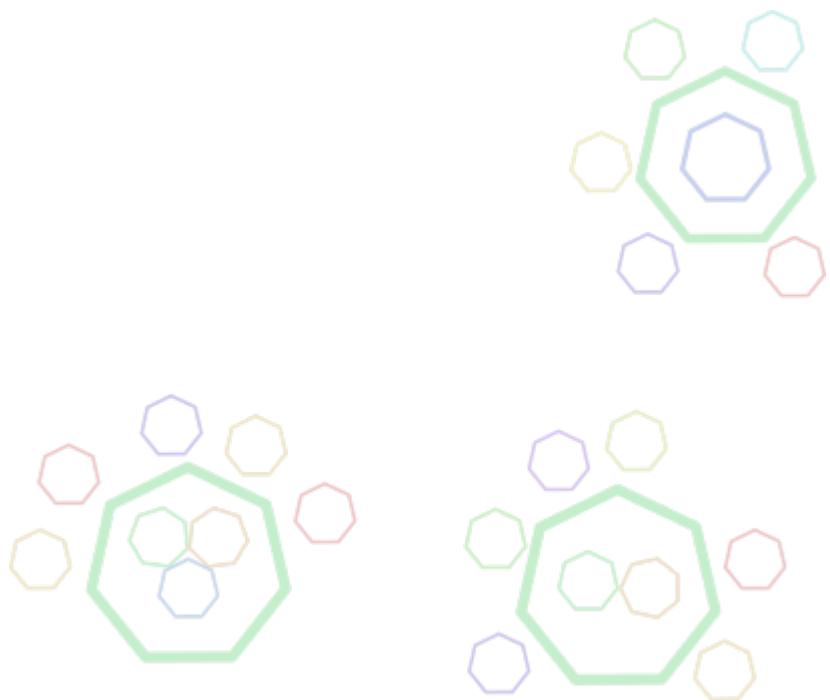


Common Pod Layout (3/3)

Adapter container

Used for supervision, act as a proxy but change data format to unify view of heterogeneous system.

- Convert metrics format
- Create metrics from API
- Unify log format



Pod Restart

Pod restart policy:

- based on the `restart_policy` parameter and status of `livenessProbe`:
 - `Always` (default)
 - `OnFailure`
 - `Never`
- applies to all containers in the *Pod*
- exponential backoff delay (10s, 20s, 40s, ...)
 - capped at 5min
 - reset after 10 min

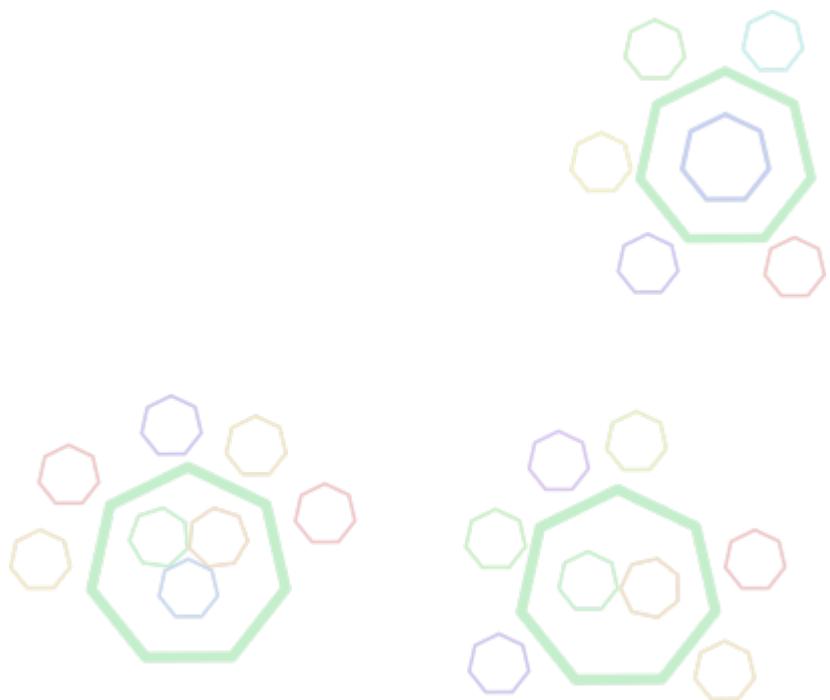


Declare Pod ports

Exposed port from containers need to be declare with the `ports` key.

- Pod can expose multiple ports
- Port can have name

```
kind: Pod
apiVersion: v1
spec:
  containers:
    - name: app
      image: company/app:latest
      ports:
        - containerPort: 80
          name: http
          protocol: TCP
        - containerPort: 9090
          name: metrics
```



Publish Pod port on host

Like with docker, you can publish some ports of a *Pod* directly on the Host.

```
kind: Pod
apiVersion: v1
spec:
  containers:
    - name: app
      image: company/app:latest
      ports:
        - containerPort: 80
          name: http
          hostPort: 80
```



Most *Pod* will use *Services* and *Ingress* to publish service.



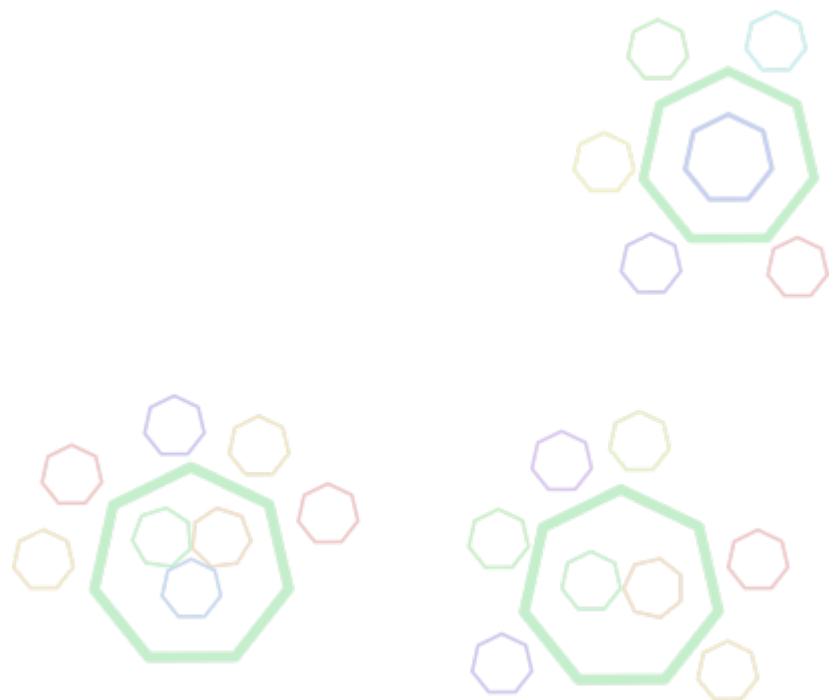
Pod Status: Liveness vs Readiness

Most of *Pods* rely on other services either in the cluster or outside. When such services fail to respond, *Pods* are not able to handle queries.

- Container should not be restarted
- But should not be used

Container is *alive* but not *ready* to use.

- `livenessProbe`: Check if *Pod* should be restarted
- `readinessProbe`: Check if *Pod* can be used as backend by a *Service*



Pod Status: Setting Probes

- Defined for each containers in `spec.containers.livenessProbe` and `spec.containers.readinessProbe`
- Several types of checks: `httpGet`, `tcpSocket`, and `exec`

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
    - name: app
      image: company/app:latest
      livenessProbe:
        tcpSocket:
          port: 8080
        initialDelaySeconds: 3
        periodSeconds: 3
      readinessProbe:
        httpGet:
          path: /health/readiness
          port: 8080
        initialDelaySeconds: 3
        periodSeconds: 3
```



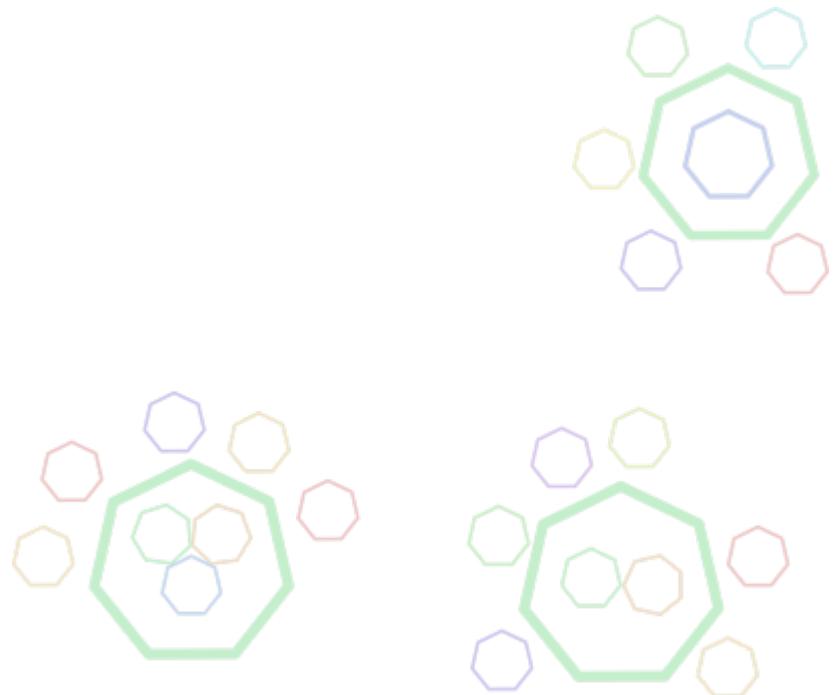
Pod Status: External Services 1

How to check health of application?

- `livenessProbe` must not rely on external services
- If probe fails, *Pod* restarts and consumes resources

Disaster scenario

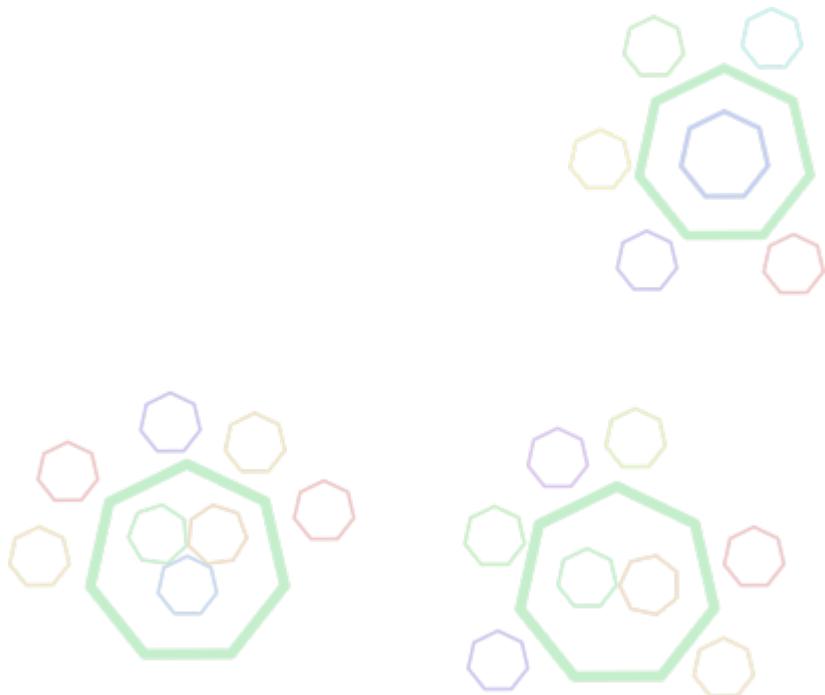
- External database is stopped for maintenance
- Almost all *Pods* in the cluster check that this database is up (liveness on external service)
- Almost all *Pods* are restarted because liveness probe fails
- ...



Pod Status: External Services 2

- Almost all *Pods* are restarted because liveness probe fails
- Almost all *Pods* are restarted because liveness probe fails
- Almost all *Pods* are restarted because liveness probe fails
- Almost all *Pods* are restarted because liveness probe fails
- Almost all *Pods* are restarted because liveness probe fails
- ...
- Almost all *Pods* are restarted because liveness probe fails
- Almost all *Pods* are restarted because liveness probe fails
- Almost all *Pods* are restarted because liveness probe fails
- Almost all *Pods* are restarted because liveness probe fails
- All resources are consumed by *Pods* restarts
- Remaining *Pods* are also impacted

Cluster performance is degraded



Pod Status: External Services 3

- Database is up and running
- Due to exponential backoff restart delay, kubernetes restarts *Pods* once every 30min
- Application stays down for 30 min even if database is up



Solution: monitor external services in Readiness endpoint, not in Liveness.



Lab 21.1: Create Pods



Objective

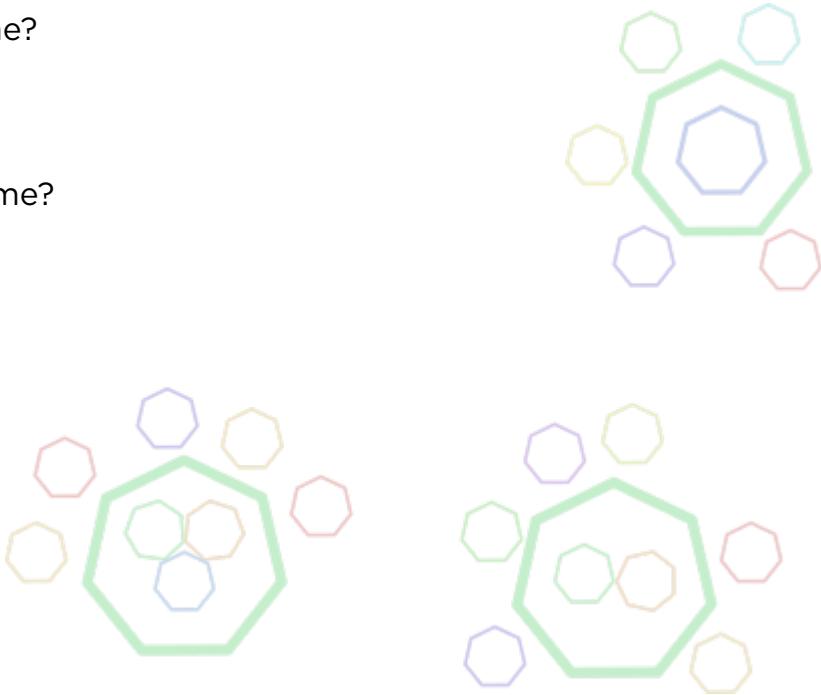
- Create and list pods

Steps

- Create a Pod with an Apache webserver
- List running Pods
- Create another Pod with nginx

Questions

- Can we run multiple versions of Nginx at the same time?
 - Yes
 - No
- How many Pods are running?
 - no Pods
 - 1-8
 - more than 8
- Can I run kubectl create several time?
 - Yes
 - No
- Can I run kubectl replace several time?
 - Yes
 - No

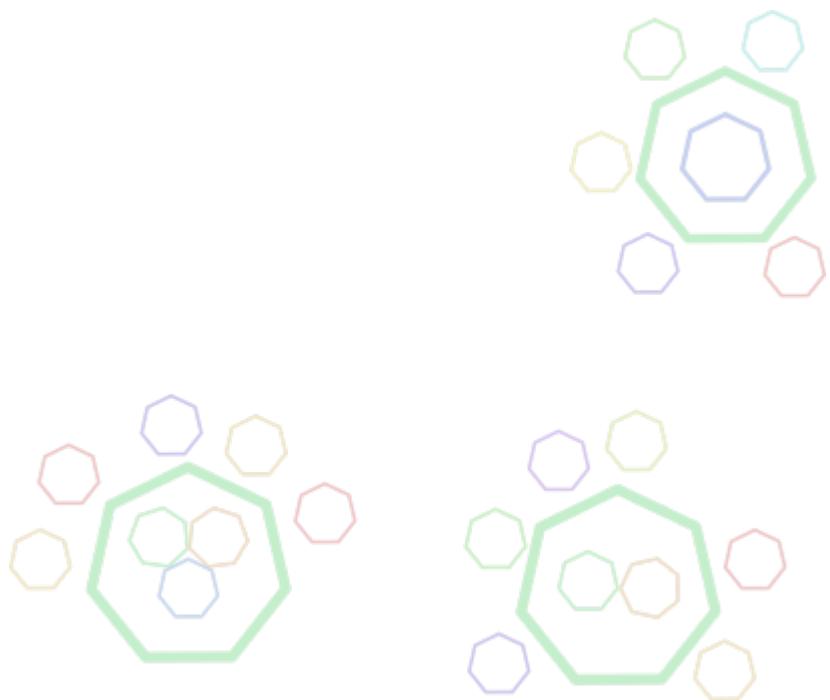


Checkpoint: Pods

- Can we avoid container restart?
 - yes
 - no

- How can we pass parameters to an application?
 - environment variables
 - entrypoint
 - command
 - init container
 - configMap
 - mounting local file

- Readiness probe can use external service?
 - yes
 - no



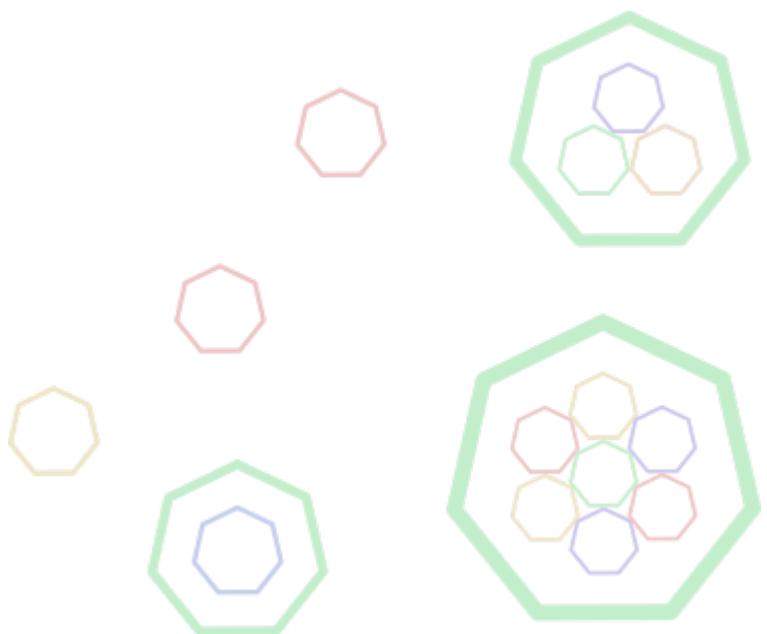
CONTROLLERS

Lesson 22: Controllers

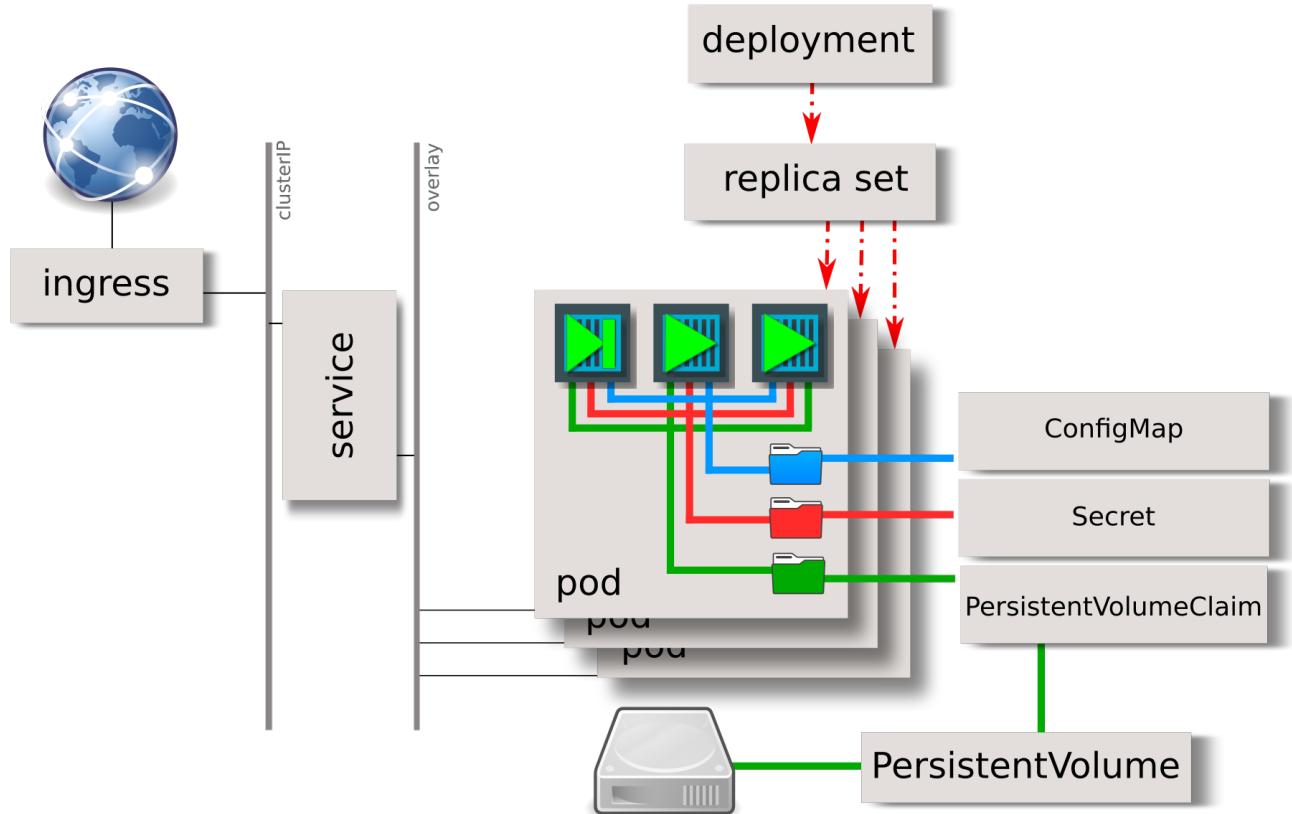
Objectives

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

- Understand how Kubernetes manages Pods
- Inspect *ReplicaSet* objects using labels and selectors
- Understand the use of *Deployments*
- Perform a rollout and rollback

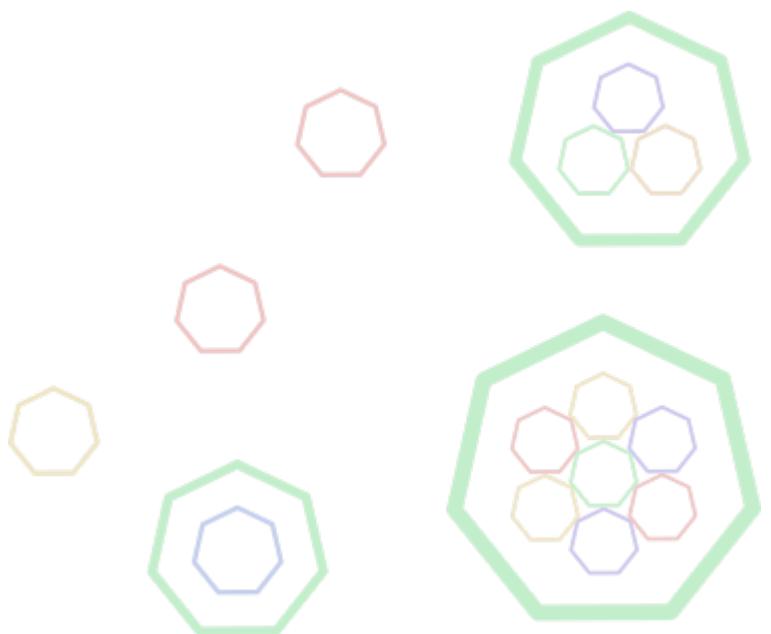


API Objects Overview



How are Pods managed?

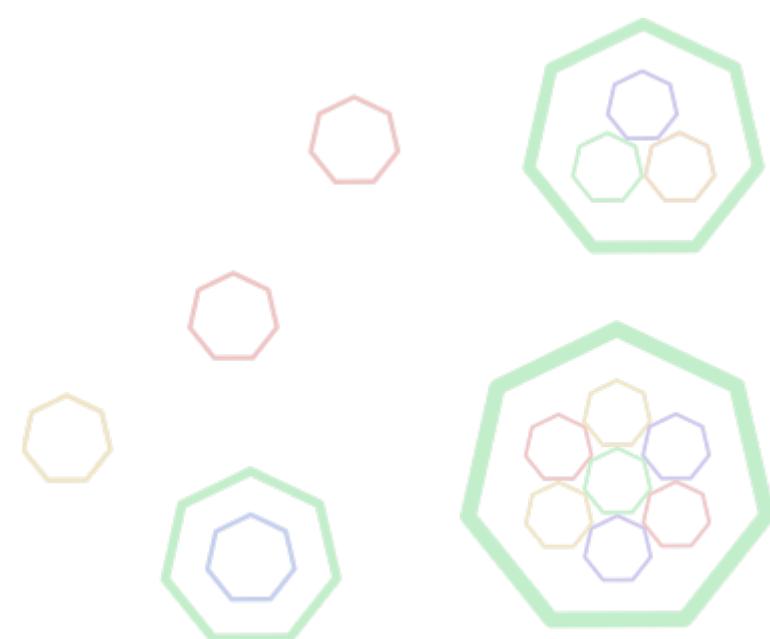
- Controllers
- Long time running with ephemeral Pods
- scale, migrate, update, schedule, orchestrate



Controllers

Several types of controllers available:

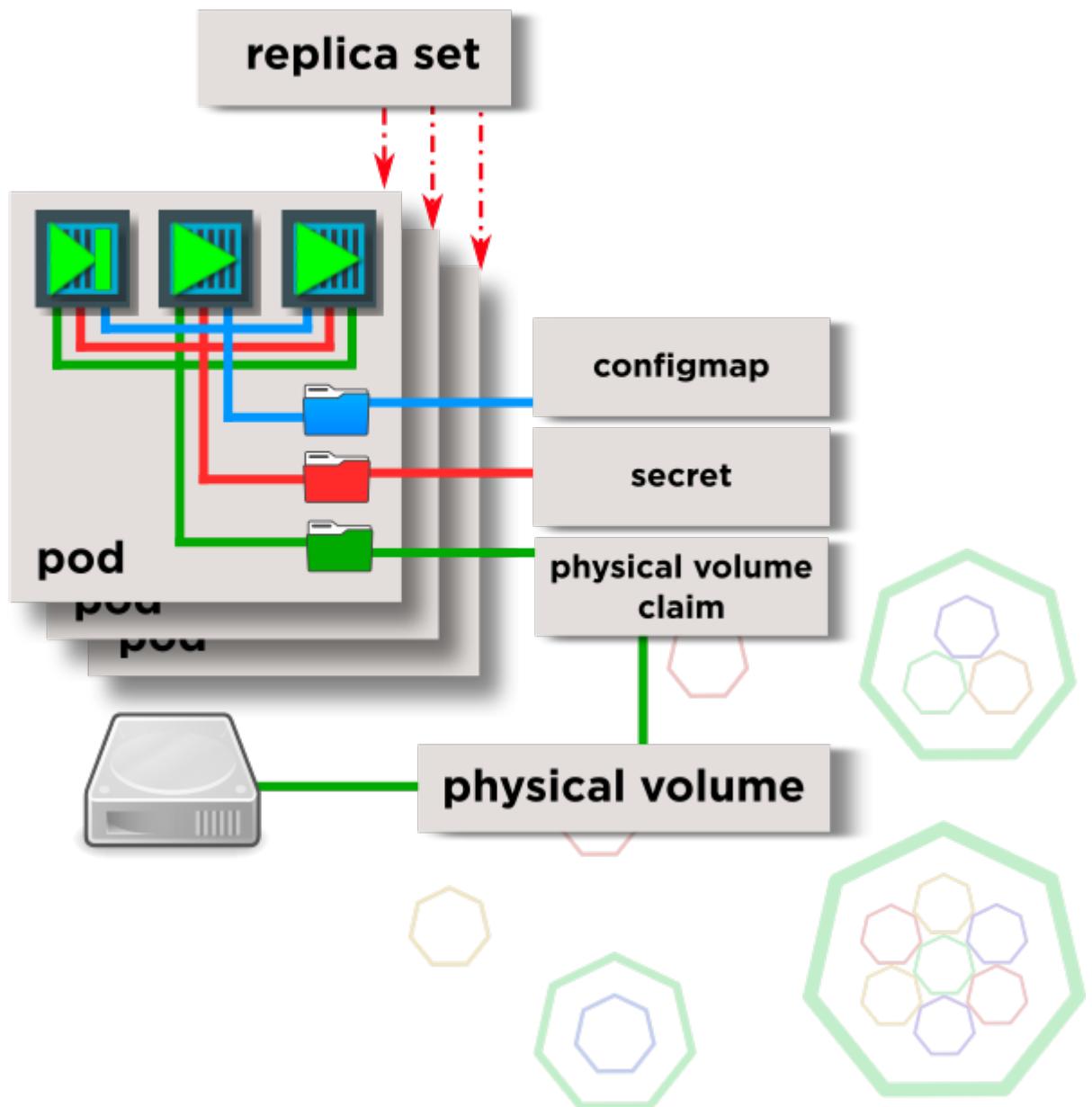
- Long term running
 - *Deployment*: Deploy Pod
 - *DaemonSet*: Run one Pod per Node
 - *StatefulSet*: For stateful apps
- Batch
 - *Job*: one time batch
 - *CronJob*: schedule batch



How to scale Pods?

You can duplicate the Pod manifest and change the Pod name but this is the purpose of *ReplicaSets*:

- Ensure specified count of Pods (maintains scaling)
- Supervise Pods: Create new Pod if one is deleted or crashed
- Use Pod template to create new Pods



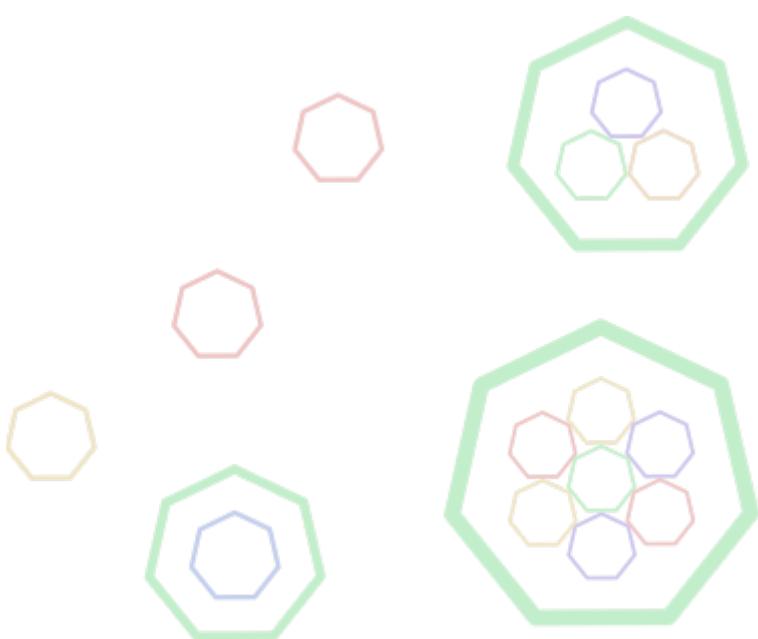
How to handle Pod modifications?

In *ReplicaSets*, Pod template is **immutable**. To change *Pod* template, you can:

- create a new *ReplicaSet* manifest and adapt *Pod* template
- Remove the old one

You can also reverse order based on the app behaviour. You may want to rollback if new configuration contains bug.

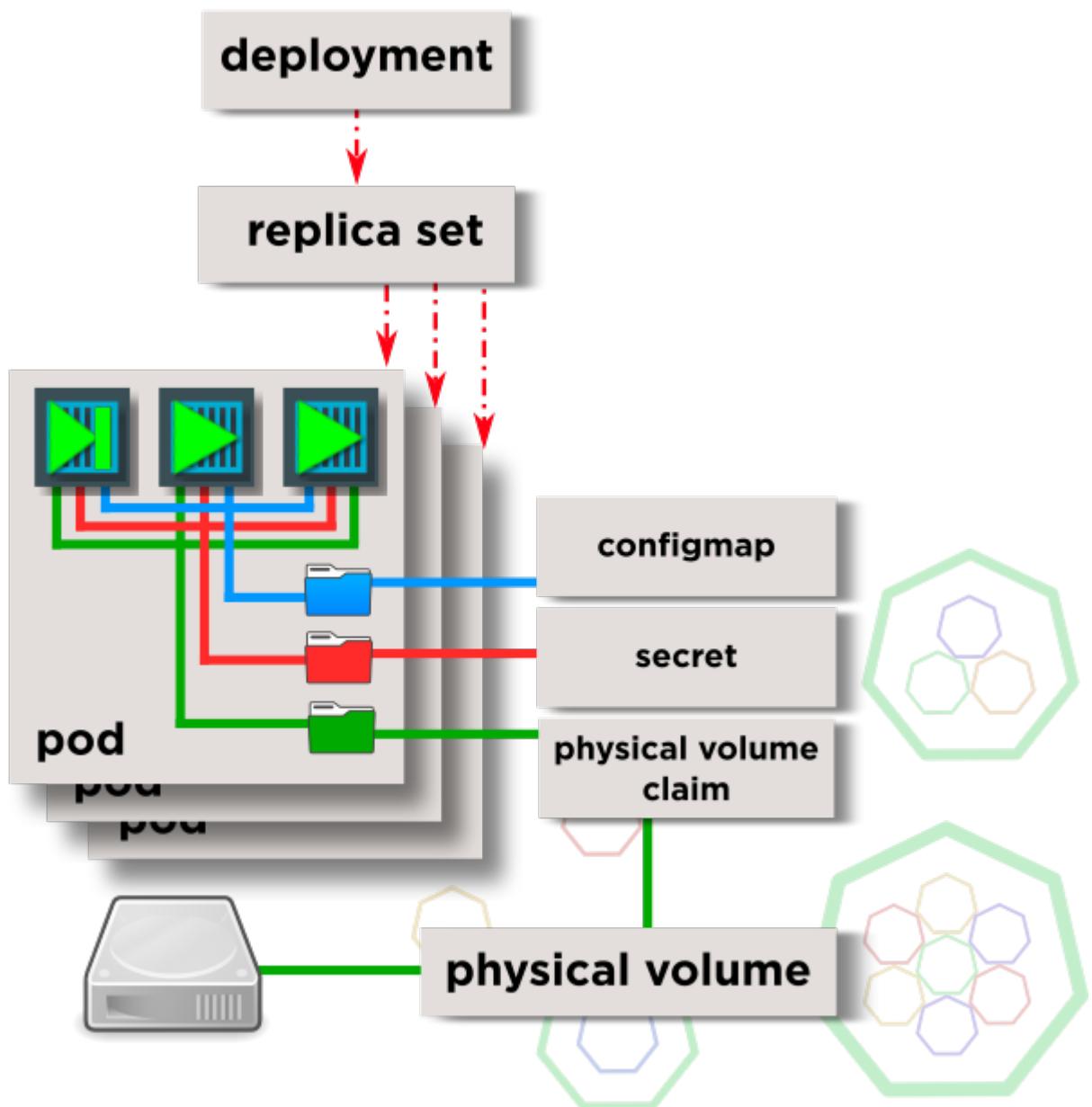
This is the purpose of *Deployments*.



Deployments

Controls *ReplicaSets* and performs transitions between them.

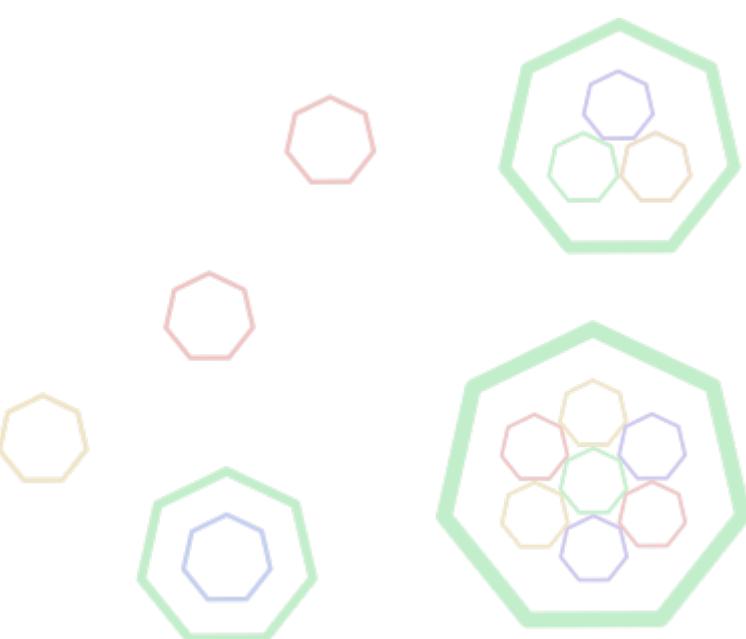
- Rollout/update ReplicaSets (adds history)
- Migrates Pods from one RS to another
- Deployments can be rolled back or paused (does not pause/stop Pods)



Deployment – Syntax

As YAML manifest:

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
  labels:
    app: apache
spec:
  replicas: 3
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
        - name: apache
          image: httpd:2.4
          ports:
            - containerPort: 80
```



Deployment – Scaling

Edit the `spec.replicas` field, either by:

- editing local manifest and applying it:

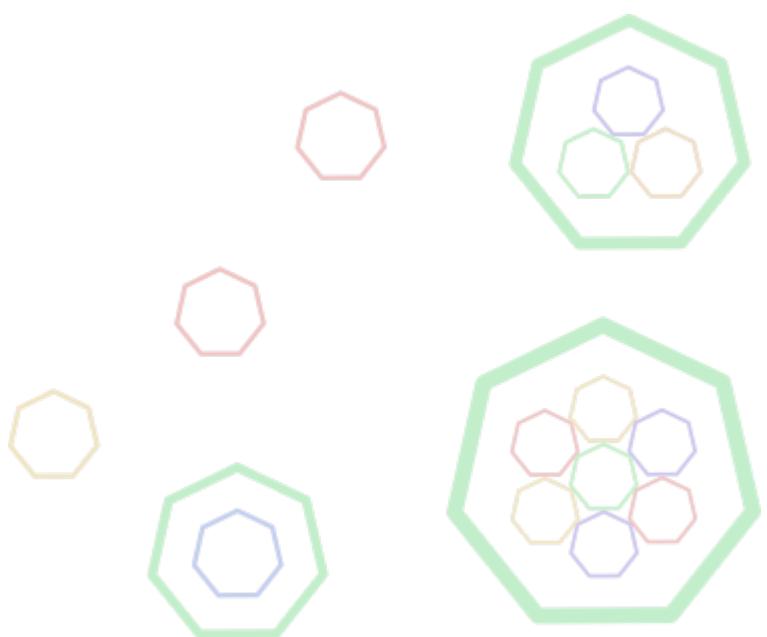
```
kubectl apply -f <manifest>
```

- editing with `kubectl edit`:

```
kubectl edit deployment <deployment name>
```

- using `kubectl scale`:

```
kubectl scale deployment <deployment name> --replicas=4
```



Deployment – Rolling update



ReplicaSets are immutable, template modifications create new ReplicaSets

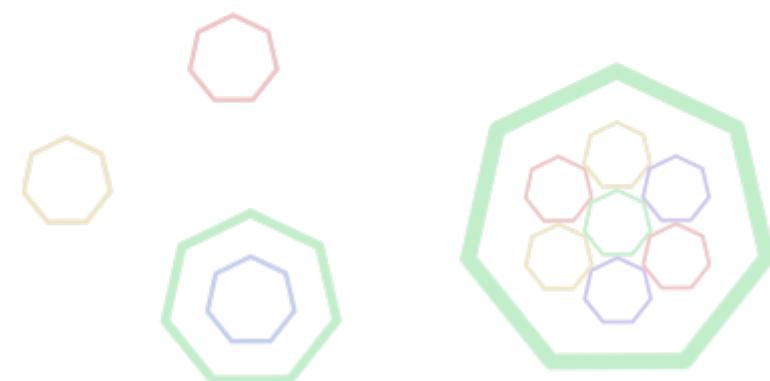
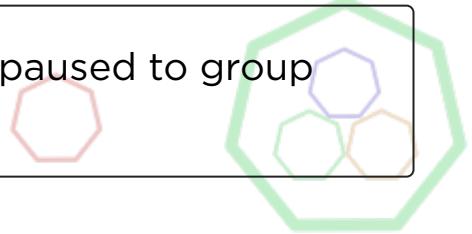
Deployments:

- Terminate old *Pods* (old *ReplicaSet*) only when new *Pods* (new *ReplicaSet*) are ready
- Keeps *Pods* count between min and max defined by:
 - `maxSurge`: supplemental *Pods* for update (default: 25%)
 - `maxUnavailable`: max unavailable *Pods* count (default: 25%)

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
  rollingUpdate:
    maxSurge: 5          # Add 5 Pods at a time max
    maxUnavailable: 10%  # No more than 10% of all Pods can be unavailable
```



Deployment rollouts can be paused and unpause to group updates



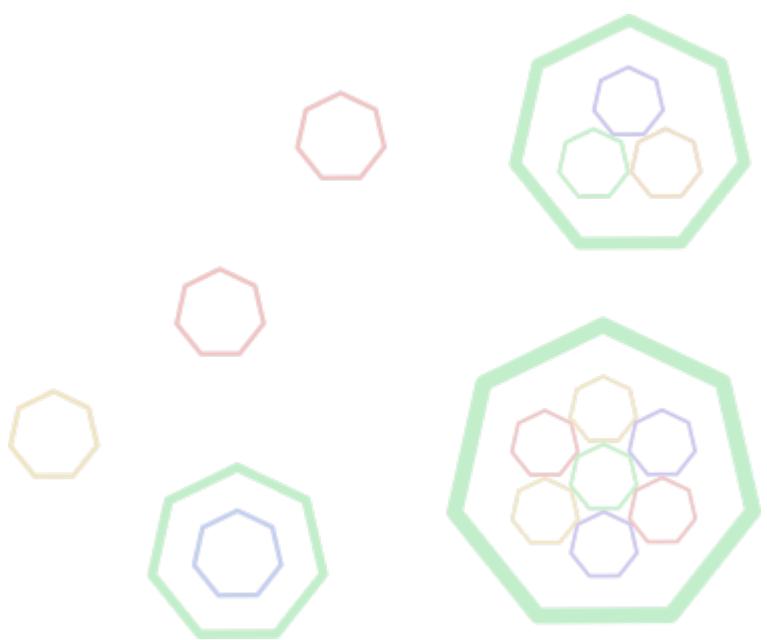
Deployment — Recreate

- For applications that require turning off old *Pods* before creating the new ones

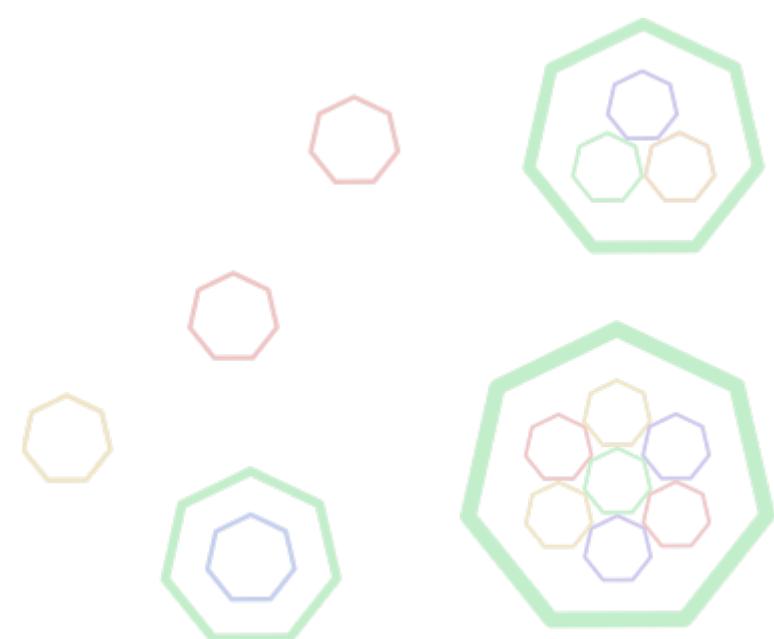
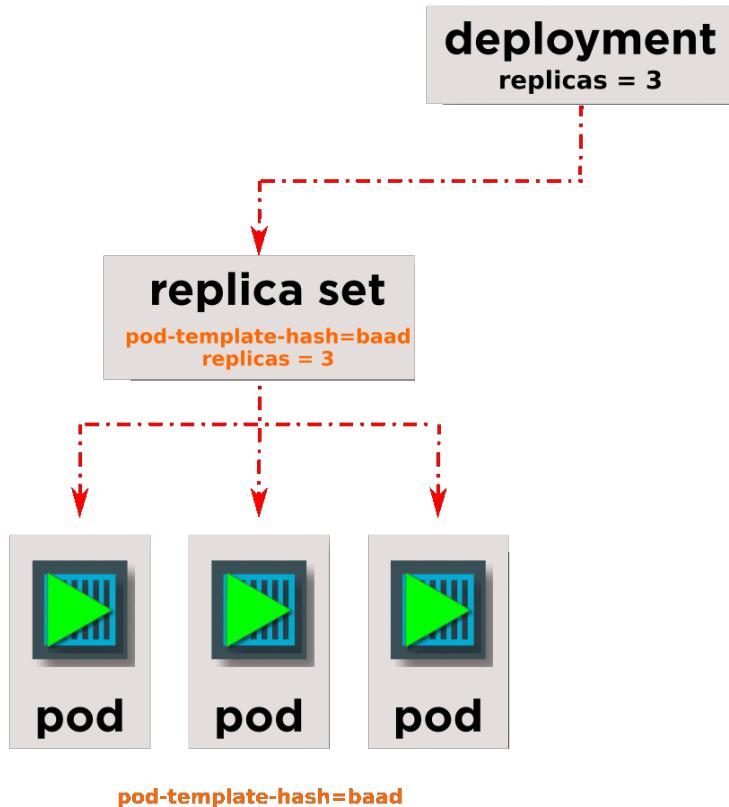
```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
spec:
  replicas: 3
  strategy:
    type: Recreate
```



The `Recreate` strategy does not guarantee a rollout without a service interruption



Deployment – Rolling update



Deployment – Rollouts & Rollbacks

Rollout status:

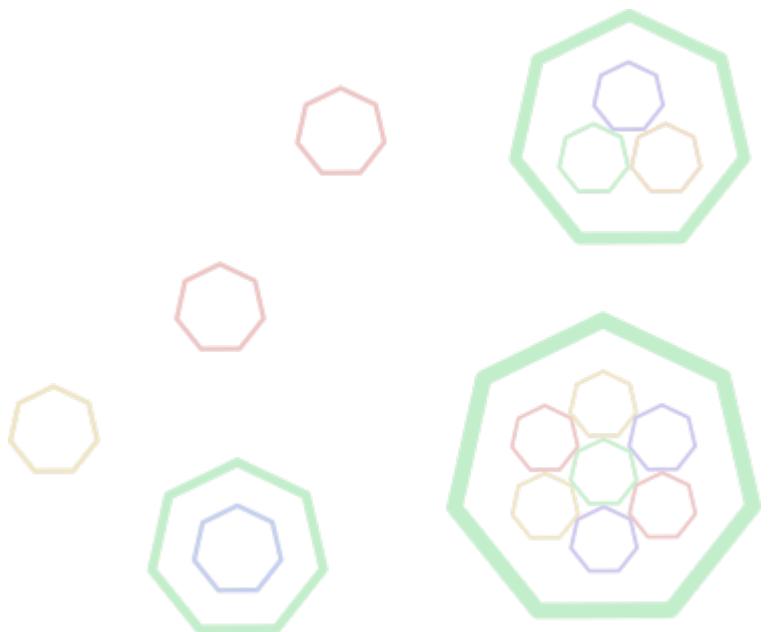
```
$ kubectl rollout status deployment apache
$ kubectl describe deployment apache
$ kubectl get rs
```

Rollout history:

```
$ kubectl rollout history deployment apache
$ kubectl rollout history deployment apache --revision=2
```

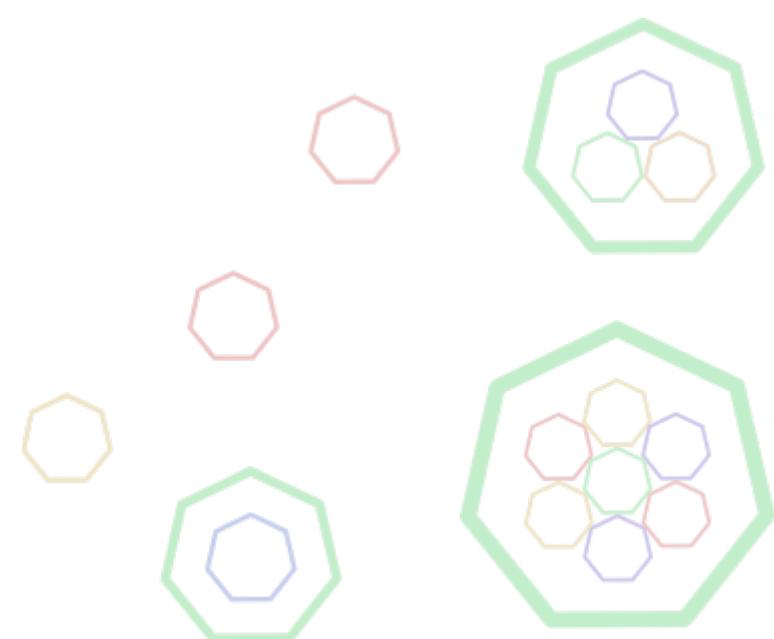
Rolling back:

```
$ kubectl rollout undo deployment apache
$ kubectl rollout undo deployment apache --to-revision=1
```



Jobs

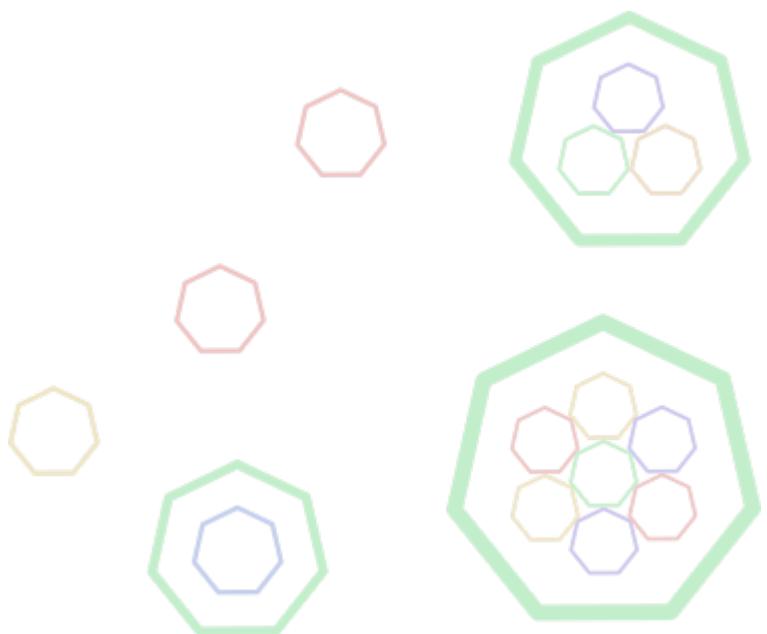
- Expected to terminate
- Launches Pod and ensures completeness
- Allows parallelism, restart
- Immutable, runs once at creation
- Container exit code sets Job status:
 - Succeeded
 - Failure



Jobs — Syntax

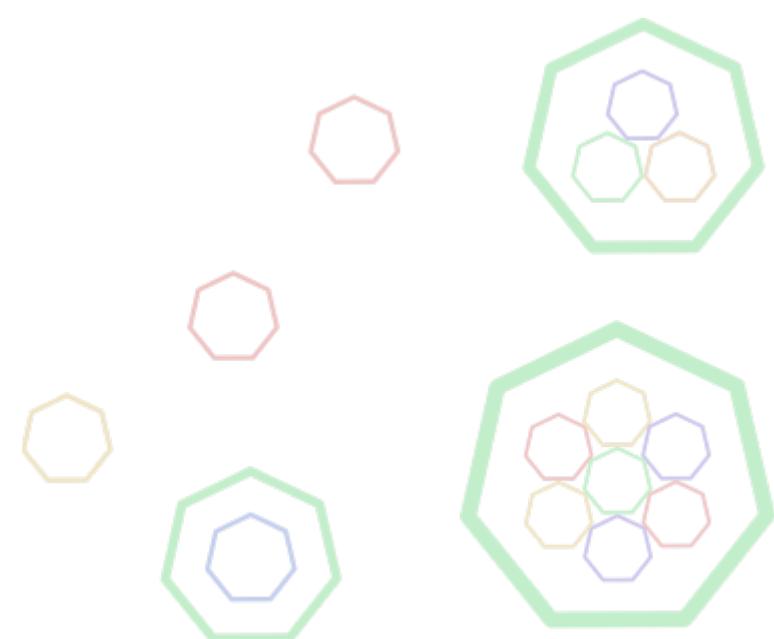
- `spec.template` has the same structure as a Pod definition
- Other `JobSpec` fields are dedicated to Jobs

```
kind: Job
apiVersion: batch/v1
metadata:
  name: generate-thumbnails
spec:
  backoffLimit: 5
  activeDeadlineSeconds: 100
  parallelism: 4
  completions: 10
  template:
    spec:
      containers:
        - name: imagemagick
          image: imagemagick
          command:
            - "mogrify"
            - "-format"
            - "gif"
            - "-path"
            - "thumbs"
            - "-thumbnail"
            - "100x100"
            - "*.jpg"
      restartPolicy: Never
```



CronJob

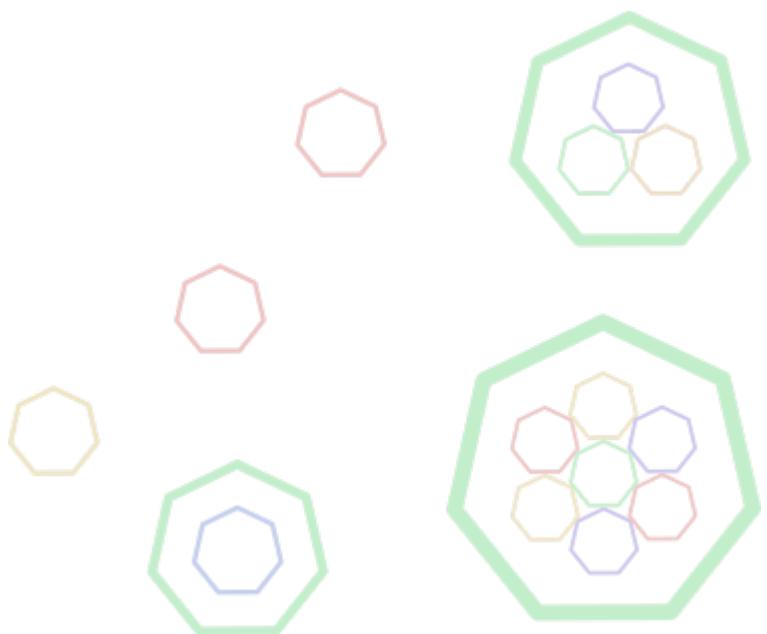
- Same as Job but runs several times: should be idempotent
- Scheduling based on cron syntax : "0 2 * * *"
- Available since Kubernetes 1.4
- Starting deadline can be configured with startingDeadlineSeconds
- `concurrencyPolicy: Allow` allows CronJob to run in parallel



CronJob – Syntax

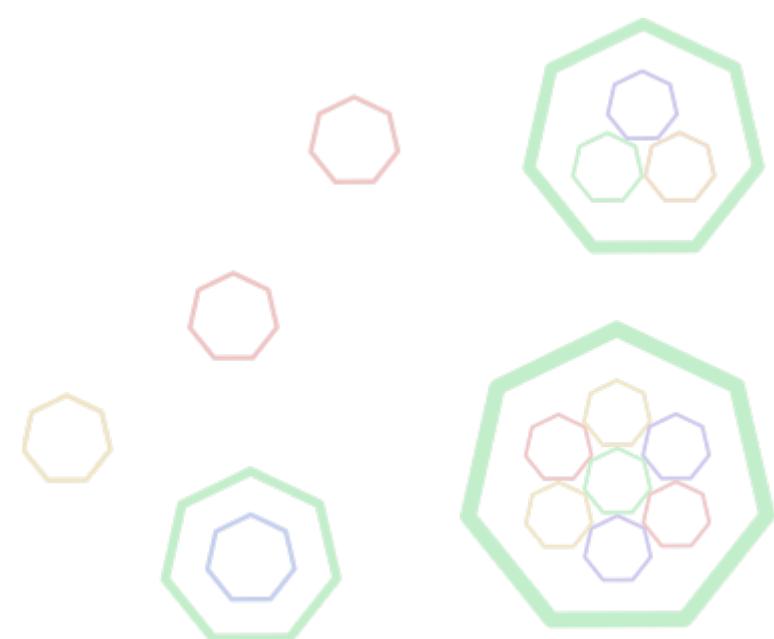
- Like Job but adds **some fields**
- CronJobs create Jobs for each execution
- For debugging purposes, the `schedule` field can be edited to force execution in 2 minutes (relative to edition time)

```
kind: CronJob
apiVersion: batch/v1beta1
metadata:
  name: generate-thumbnails
spec:
  schedule: "10 * * * *"
  startingDeadlineSeconds: 100
  successfulJobsHistoryLimit: 10
  failedJobsHistoryLimit: 10
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: imagemagick
              image: imagemagick
            command:
              - "mogrify"
              - "-format"
              - "gif"
              - "-path"
              - "thumbs"
              - "-thumbnail"
              - "100x100"
              - "*.jpg"
  restartPolicy: Never
```



DaemonSets

- Like `Deployment` but ensures one Pod per node
- Not expected to terminate
- Scheduling is Node oriented instead of resource oriented
- Usage: Collectd, Prometheus, Glusterd

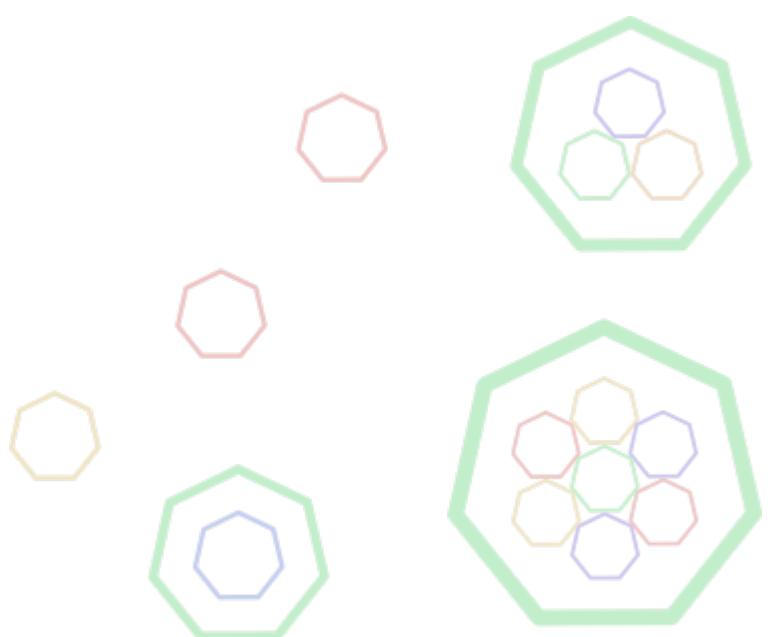


StatefulSet — Principle

- Stateful and distributed application
- Should support failures:
 - Pods created by StatefulSet can crash (node removal, hardware failure)
 - Scaling may require unavailable resources



It is recommended to avoid deploying Stateful distributed applications as containers altogether as their management is complex.

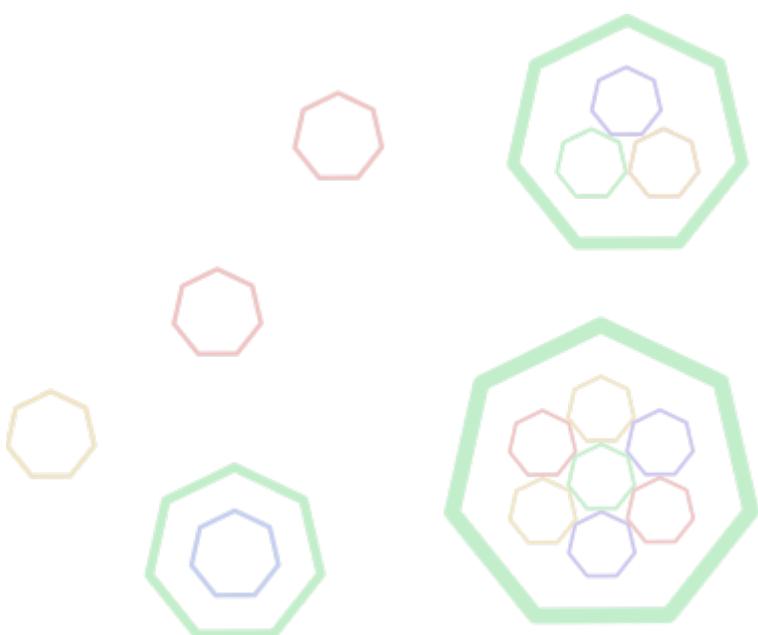


StatefulSet – Implementation

- Pods have a persistent order: app-1, app-2, ...
- Each Pod receives a dedicated volume
- Volumes are not deleted upon scaling down
- So data are persisted after scaling down and up



StatefulSets requires headless Services



Lab 22.1: Create Deployments



Objective

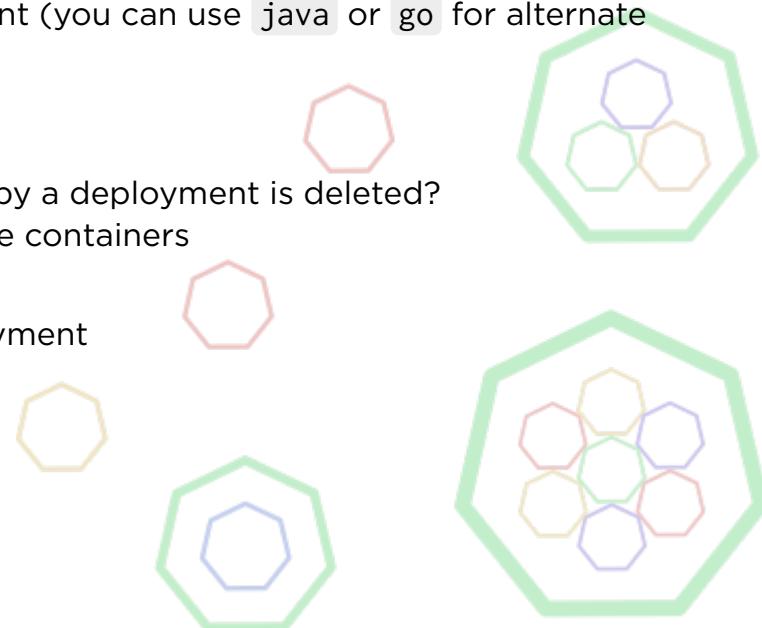
- Create a deployment for backend using `ghcr.io/camptocamp/course_docker_backend:python`
- Scale it up and down
- Delete pods, what happens?
- Check the application
- Rollout of a new version

Steps

- Cleanup old Pods
- Create a Deployment for backend
- Increase replica count
- Delete one Pod created by the Deployment
- Set up a Port Forward to a container (port `8080`)
- Check the application in a web browser
- Modify the image tag in the Deployment (you can use `java` or `go` for alternate versions of the app)

Questions

- What happens when a pod managed by a deployment is deleted?
 - Depends on the `restart_policy` on the containers
 - A new Pod is created
 - Administrator need to rollout Deployment



Horizontal Pod Autoscalers

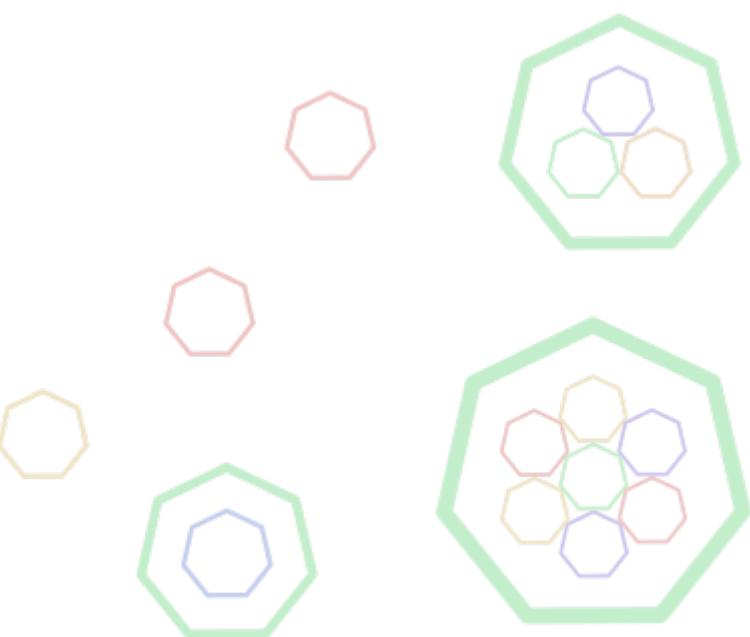
Automatically controls a Deployment's replicas based on metrics.

Create from command line:

```
$ kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

As YAML:

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
    target:
      type: Utilization
      averageUtilization: 50
```



Checkpoint: Controller

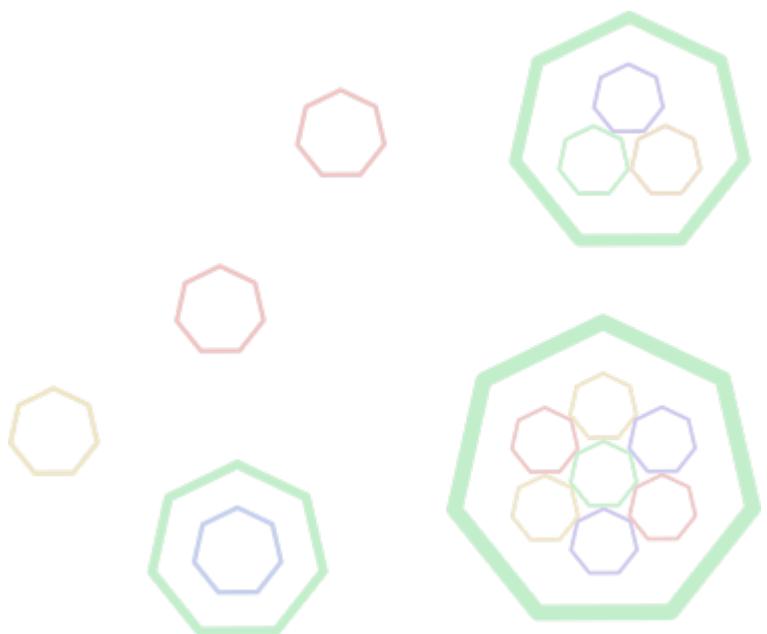
What object would you use to run a process that...

- generates thumbnails for pictures every night?
 - Job
 - CronJob
 - Deployment
 - DaemonSet
 - StatefulSet

- generates cache for static files?
 - Job
 - CronJob
 - Deployment
 - DaemonSet
 - StatefulSet

- runs a PostgreSQL database with a single "main" ?
 - Job
 - CronJob
 - Deployment
 - DaemonSet
 - StatefulSet

- runs a PostgreSQL database cluster with one "main" and a variable count of replicas?
 - Job
 - CronJob
 - Deployment
 - DaemonSet
 - StatefulSet



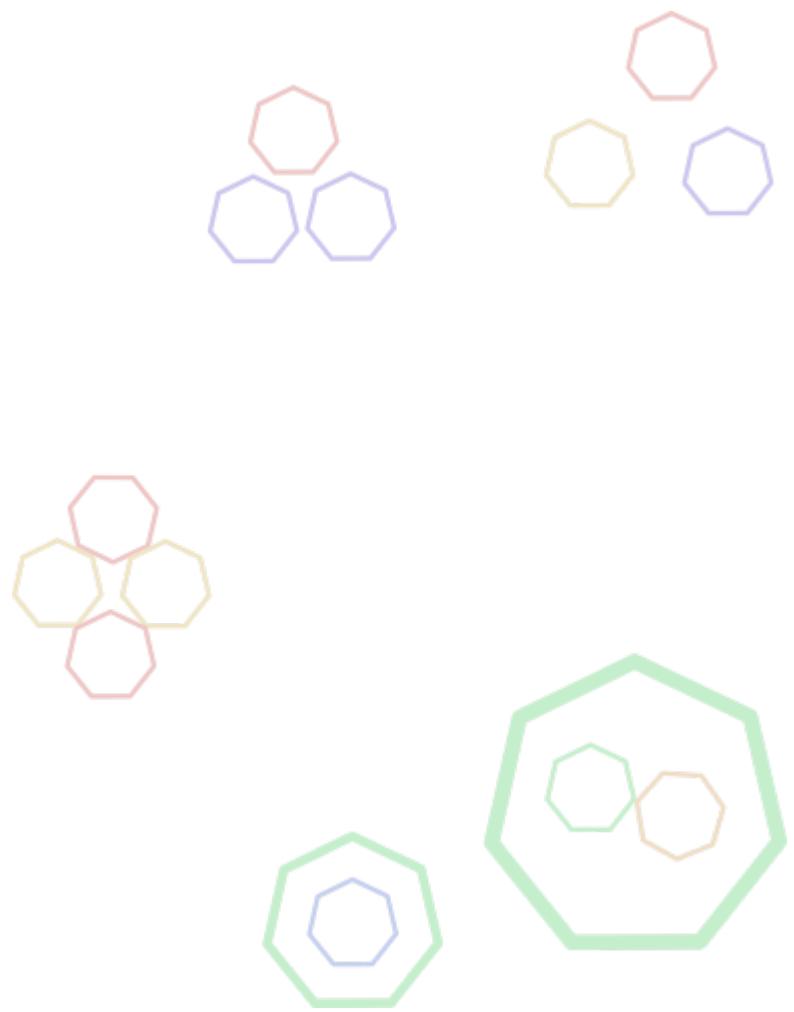
SERVICES

Lesson 23: Services

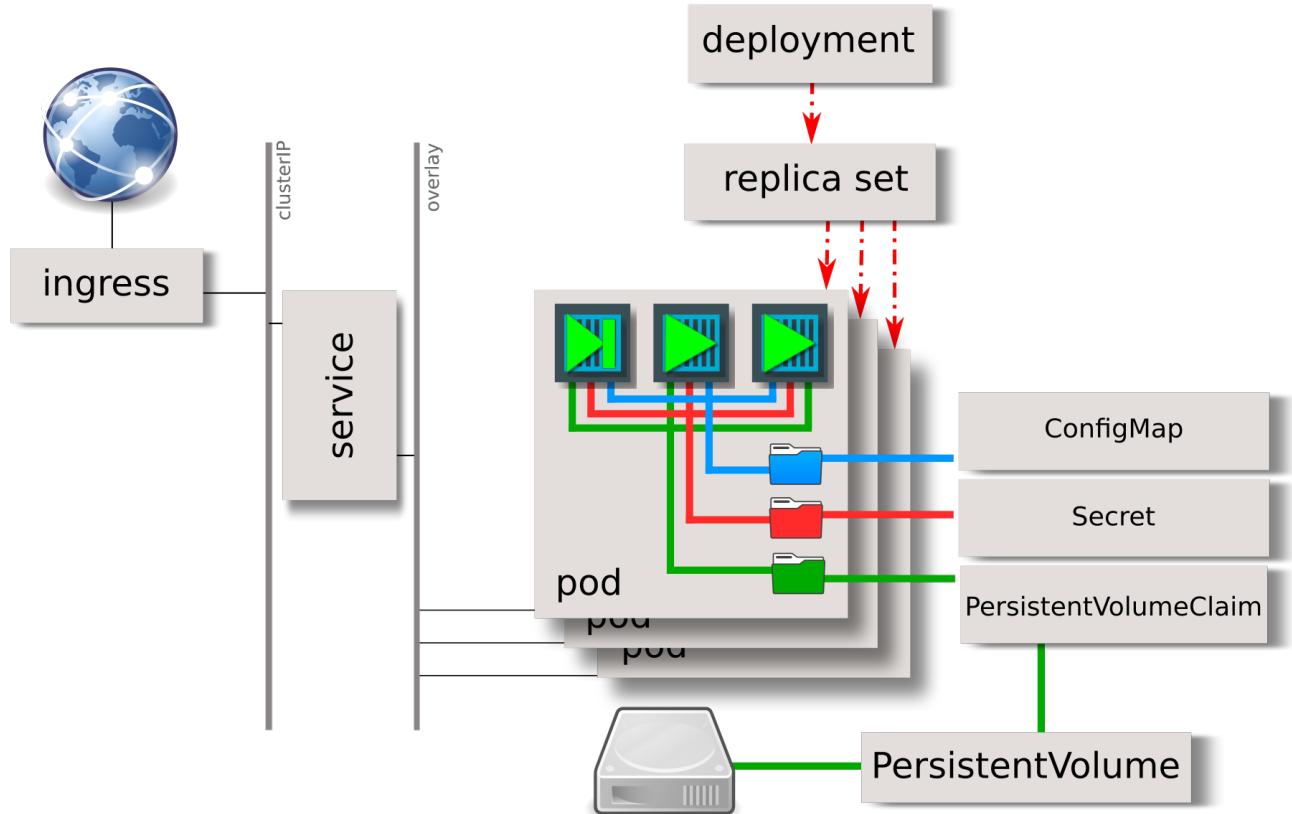
Objectives

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

- Understand the differences between the different kinds of services
- Understand network principles in Kubernetes
- Use selectors to link objects

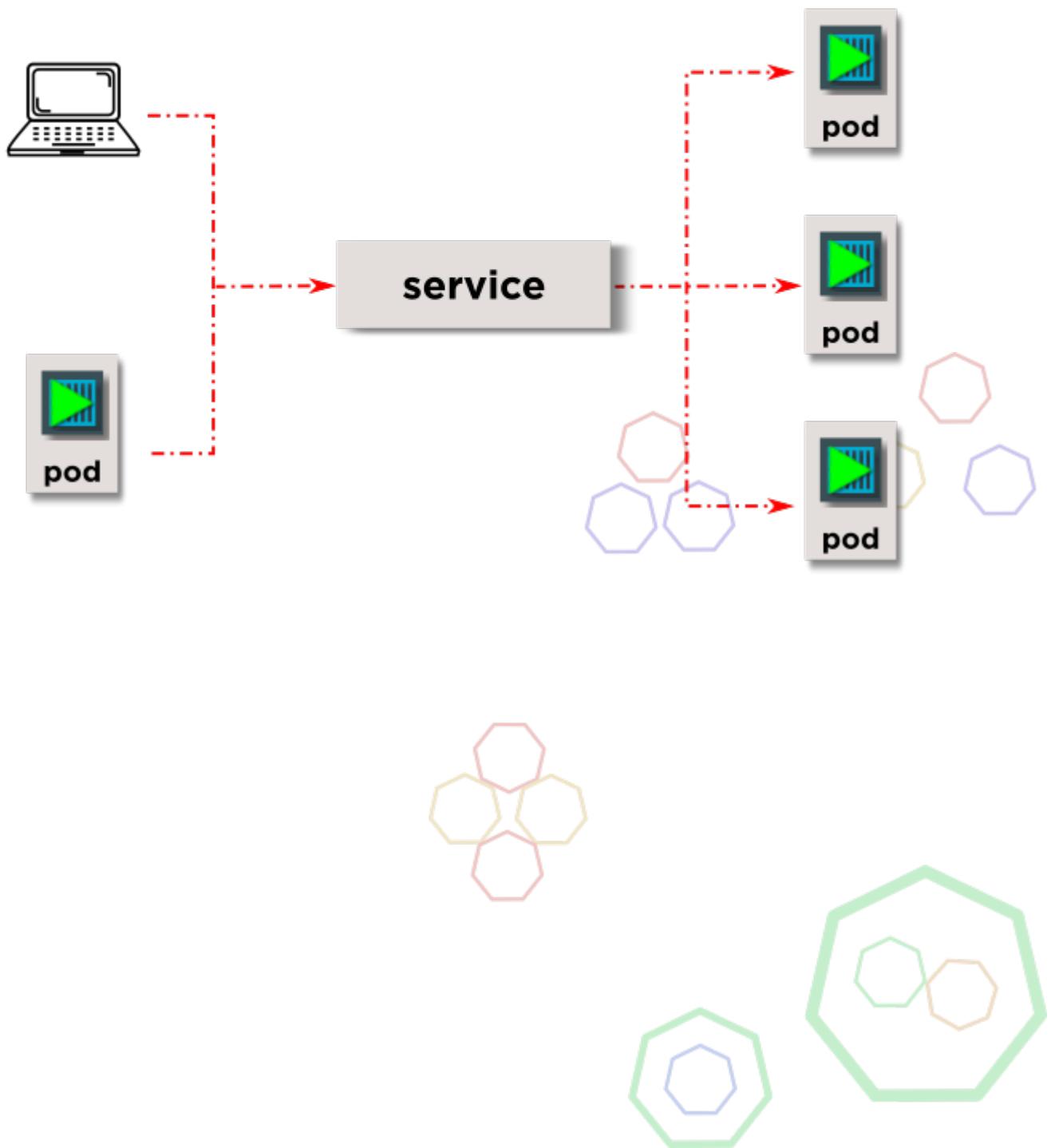


API Objects Overview



Services

- Services abstract Pods
- Long term static Endpoint for ephemeral Pods
- load balance traffic to Pods



Services – Active TCP/UDP Proxy

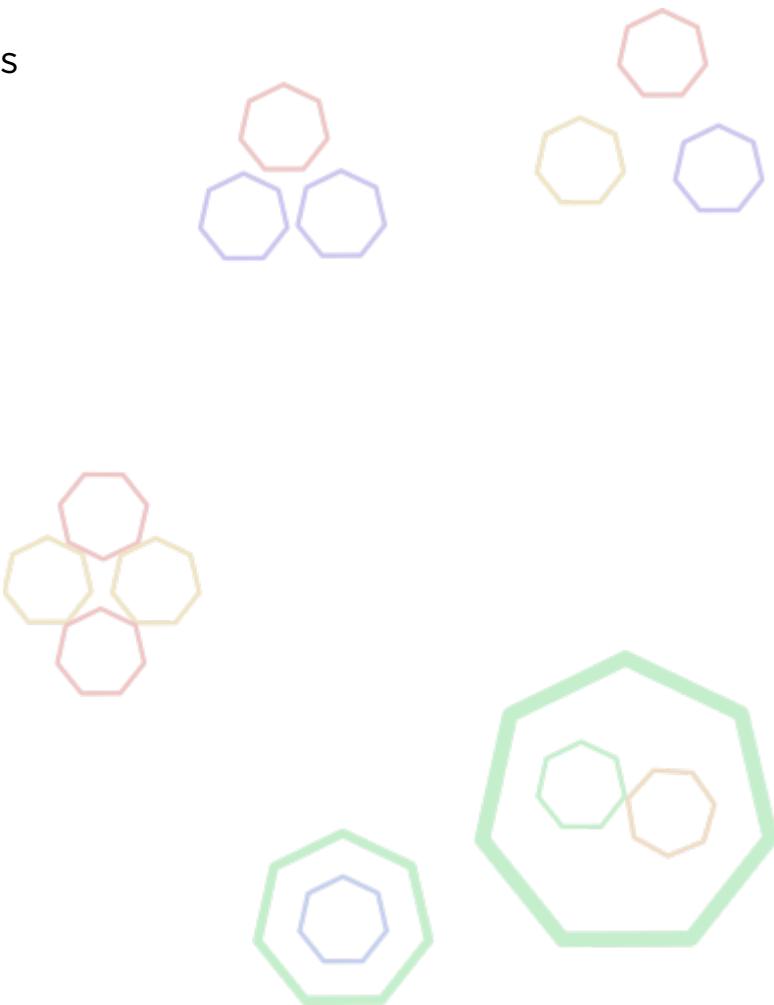
- 3 cumulative levels of services:
 - ClusterIp : Internal usage. Default type.
 - NodePort : Publish service on each host
 - LoadBalancer : Provision Layer 4 LB from IaaS



Layer 7 HTTP proxies are implemented using the Ingress Object

- Other usages:

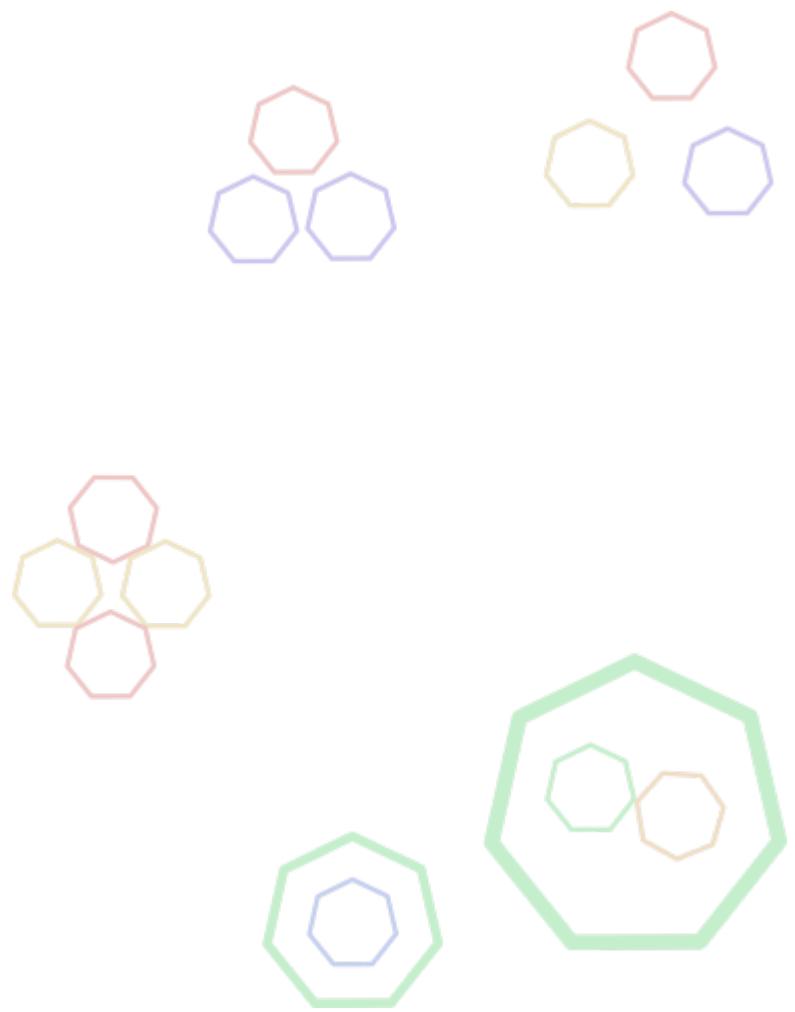
- ExternalName : DNS alias to external services (database, smtp, ...)
- manually managed Endpoints



No DNS round-robin

Kubernetes doesn't use DNS round-robin because:

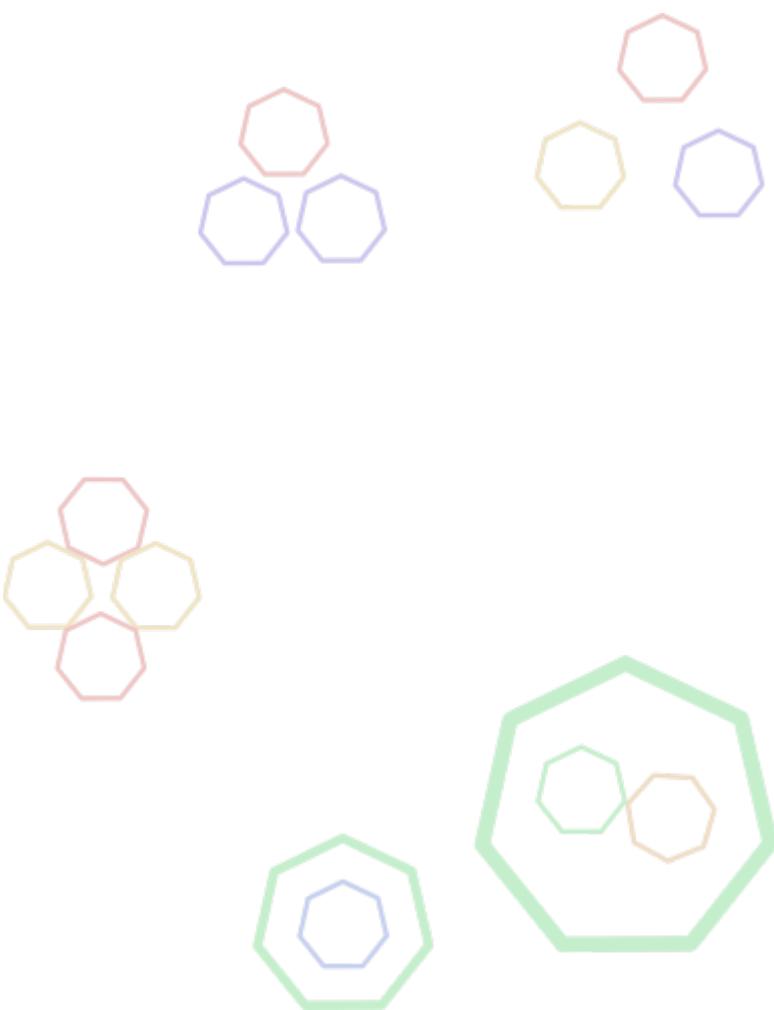
- Applications and libraries do not always honor TTL
- Some applications cache DNS results
- Requesting DNS before each request consumes resources
- Exception: Headless Service (`ClusterIP: none`)



Service Ports

- `port` and `targetPort` fields control port behaviour
- `targetPort` can refer to a named port configured on Pods (e.g. `http-alt: 8080`)

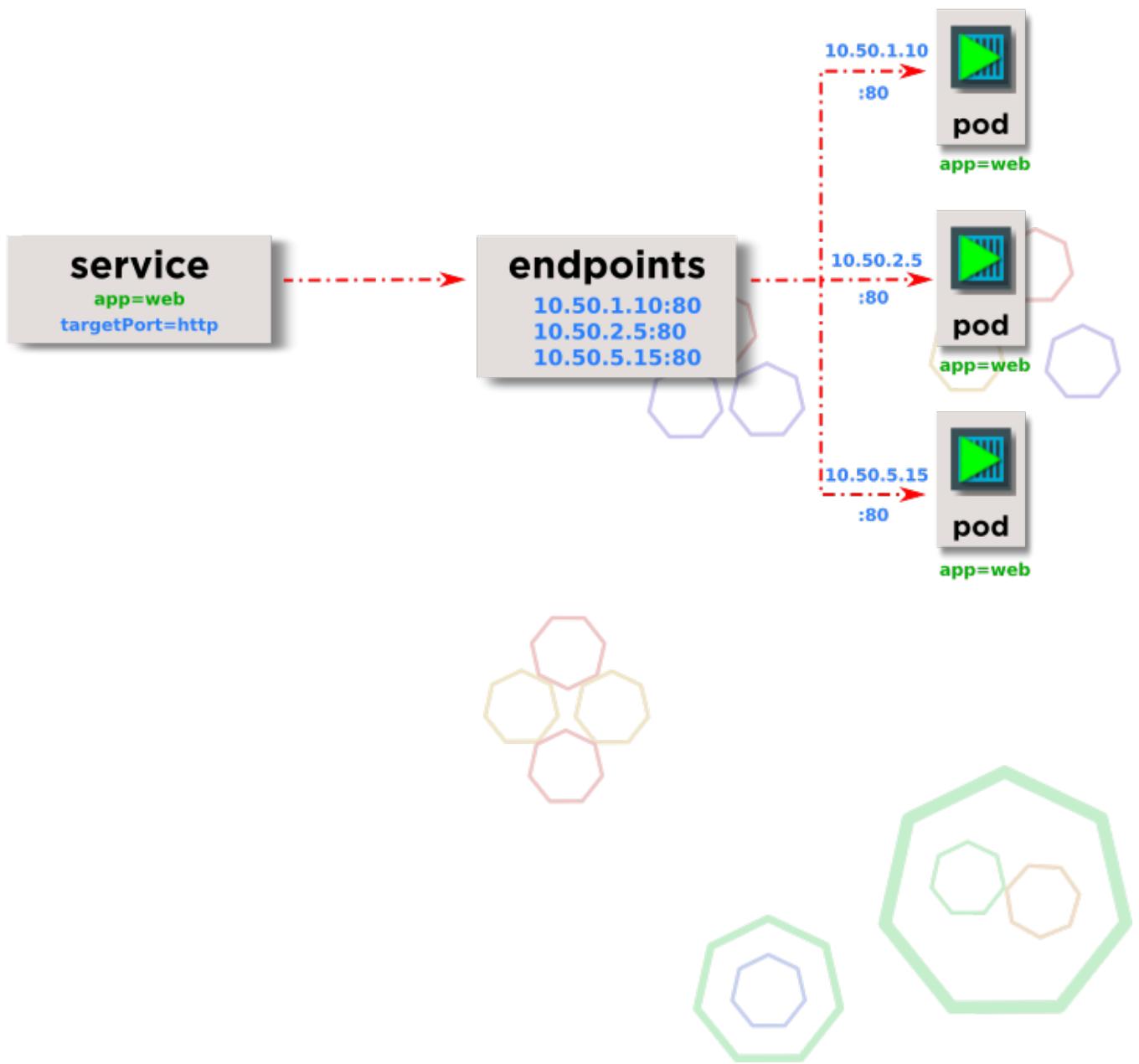
```
kind: Service
apiVersion: v1
metadata:
  name: apache
spec:
  selector:
    app: apache
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
  - name: https
    protocol: TCP
    port: 443
    targetPort: https
```



Endpoints

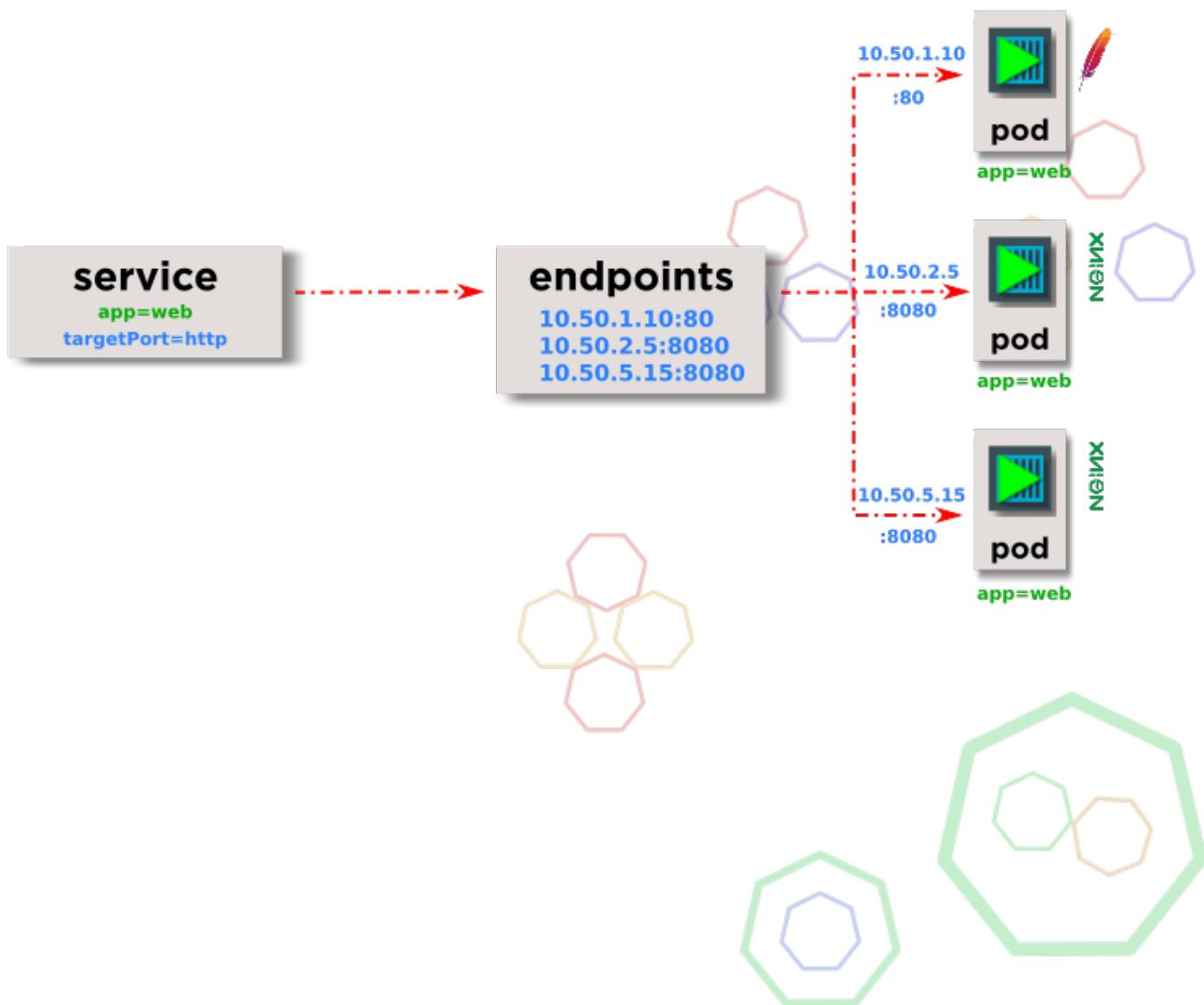
Service Endpoints are:

- a set of Pod addresses and ports
- created by Services
- populated with the results of evaluation of Label selectors
- managed manually if no selector is defined



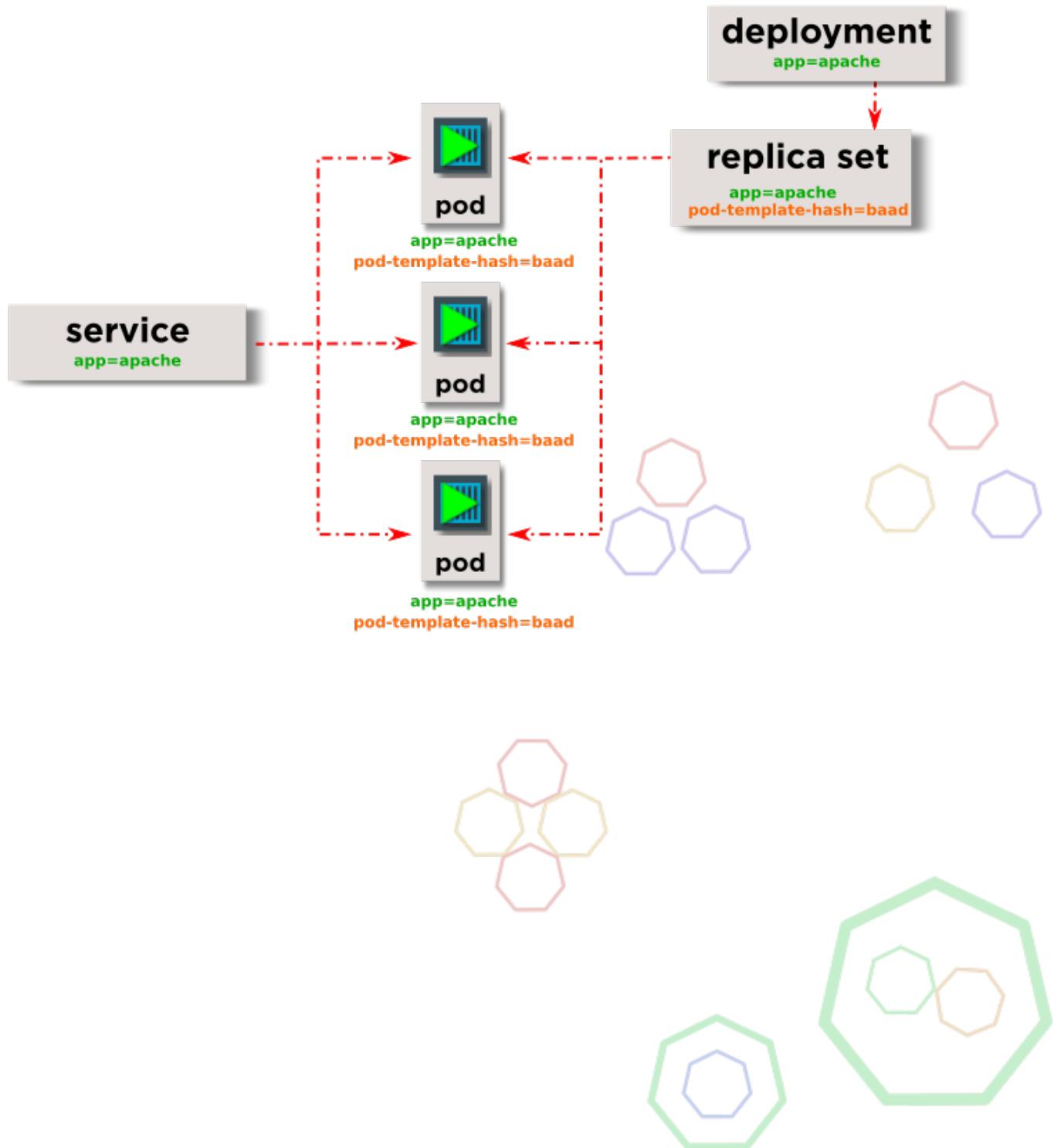
Service Port: Variable targetPort

- Named ports can point to different port numbers:
 - `http` points to 80 for Apache
 - `http` points to 8080 for Nginx
 - services are configured with `targetPort: http`
 - Usage : migration to new version with port change
- When defining multiple ports, you need to add name on port: `spec.ports.name` on Service object



Selector Overlap

- Service and Controller should use the same selector for one micro service
- Selector must not overlap between Controllers



Link with Workload

- Kubernetes uses Labels to map Services to Pods
- the `selector` field sets Labels to identify target Pods

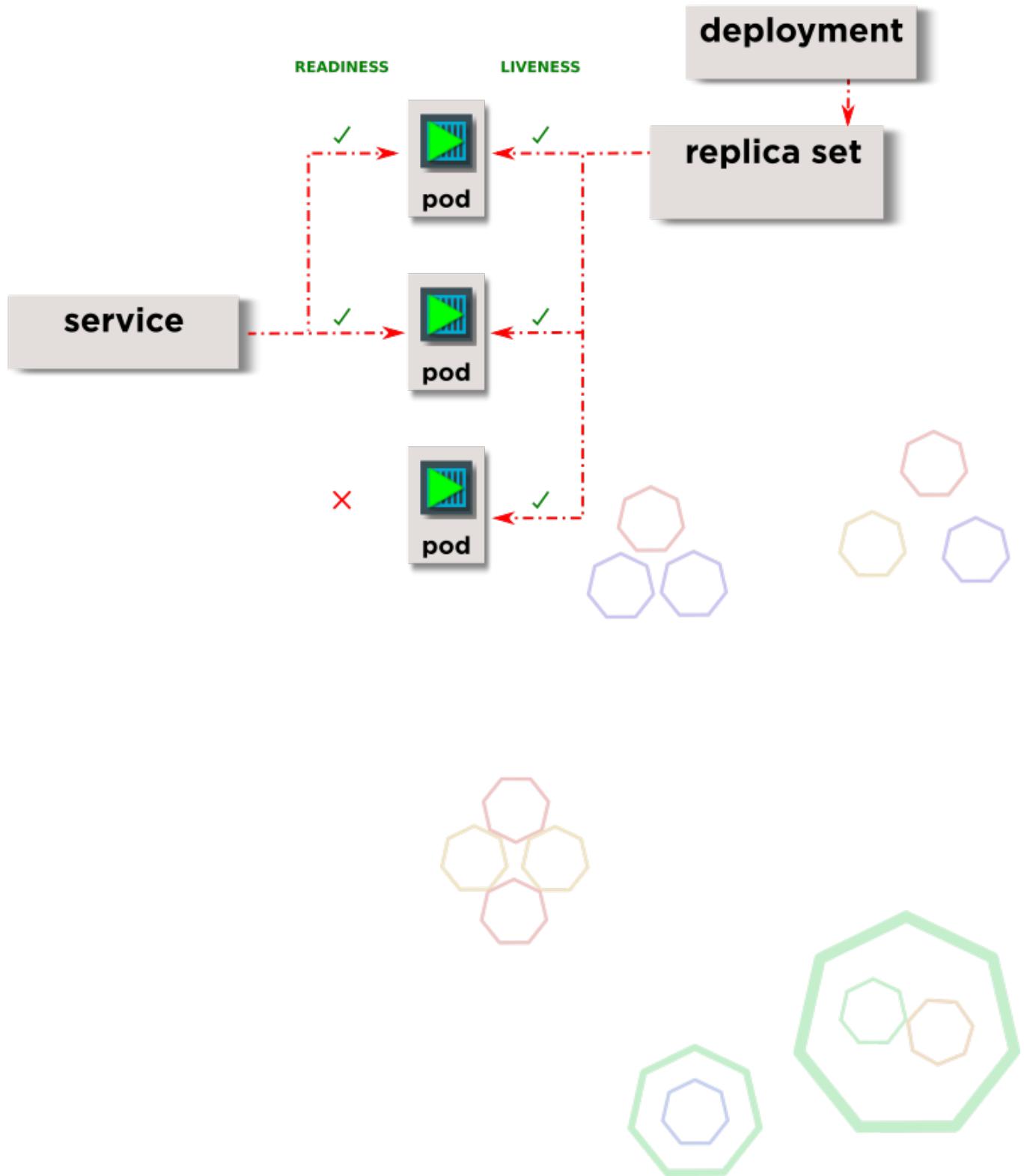
```
kind: Service
apiVersion: v1
metadata:
  name: apache
spec:
  selector:
    app: apache
```

```
kind: Pod
apiVersion: v1
metadata:
  name: apache
  labels:
    app: apache
spec:
  containers:
    - name: apache
      image: httpd:2.4
```



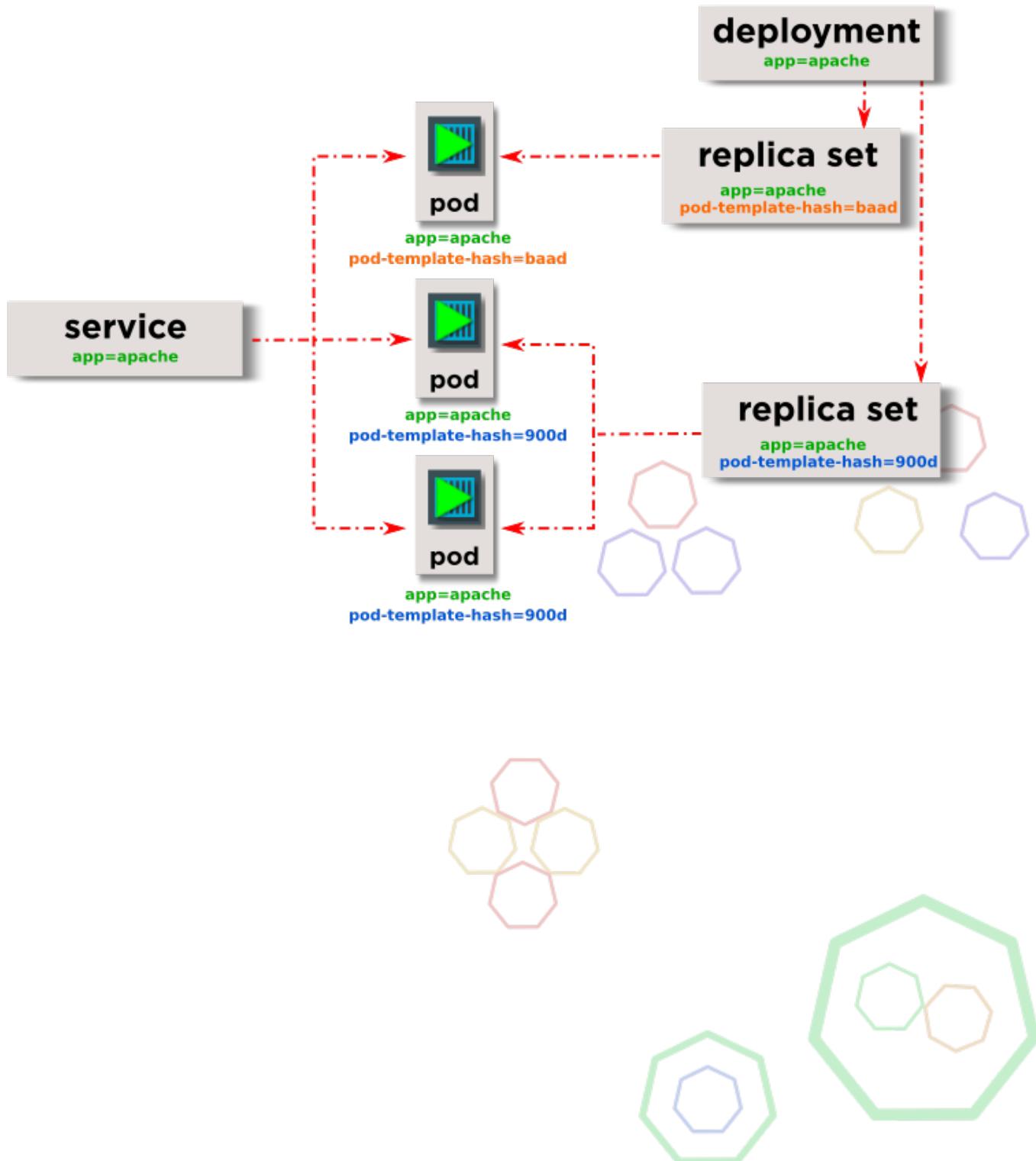
Service & Probes

Readiness probes are used to add or remove pods from the load-balancer



Service Continuity during a Rollout

- Service ignores the `pod-template-hash` label
- The service is not interrupted during the rollout

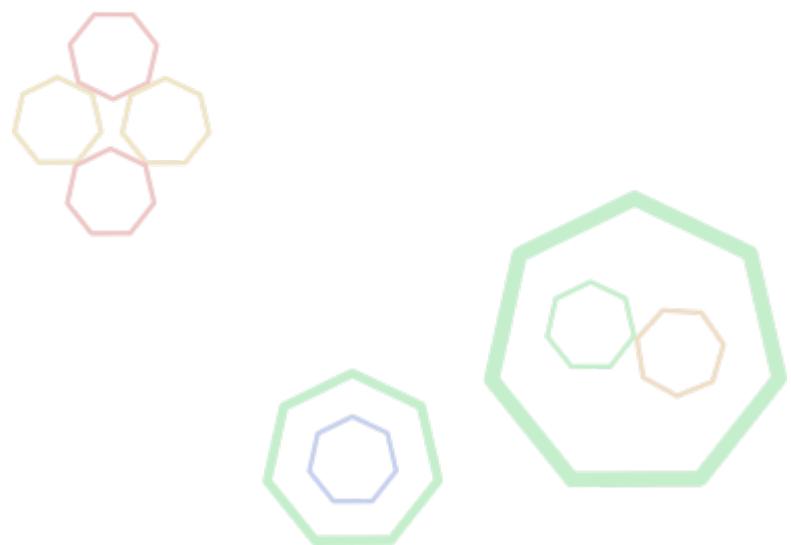


Selector Usage – API objects

Services and Deployment use selectors:

```
kind: Service
apiVersion: v1
metadata:
  name: apache
spec:
  selector:
    app: app1
    release: test
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
```

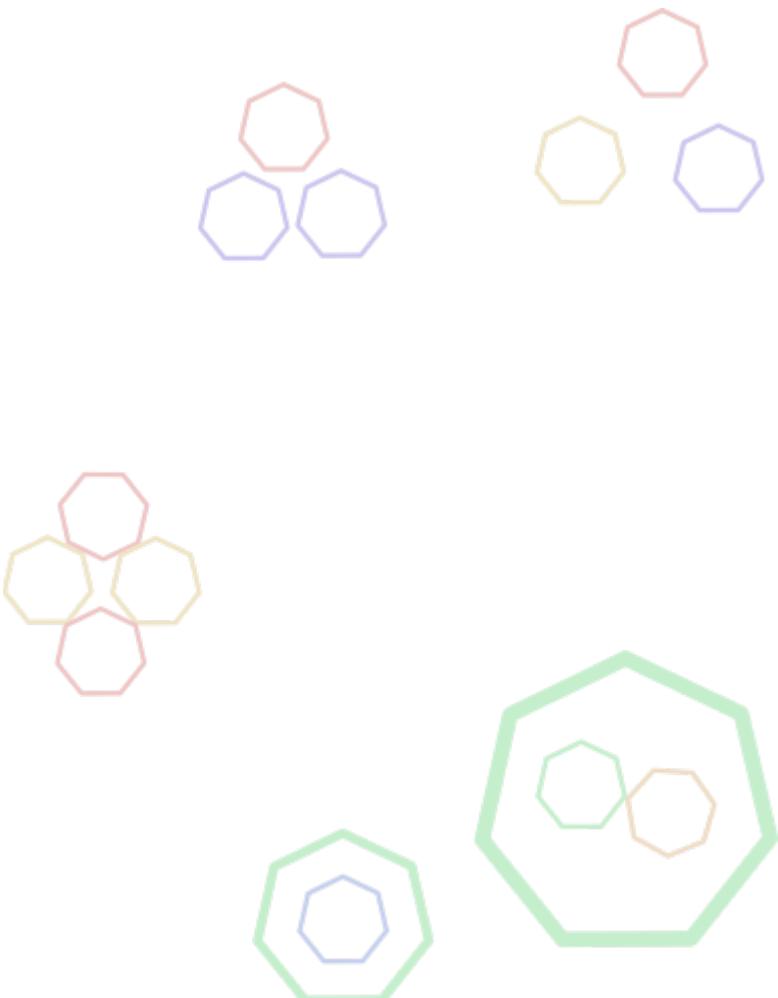
```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
spec:
  selector:
    matchLabels:
      app: app1
      release: test
  template:
    metadata:
      labels:
        app: app1
        release: test
  spec:
    containers:
    - name: apache
      image: httpd:2.4
```



Selector Usage — kubectl

- Multiple rules ANDed with a comma
- Mix equality and set-based selectors:

```
kubectl get all -l 'app=apache, tier in (frontend, backend)'
```



Service Discovery DNS

- DNS records published for each service
- name of service can be resolved in same namespace
- to access other namespace services and suffix with namespace
- `apache.dev` resolves to the `apache` service in the `dev` namespace

```
/var/cache/showoff # getent hosts argocd-server.argocd.svc.cluster.local  
172.20.238.172    argocd-server.argocd.svc.cluster.local  argocd-  
server.argocd.svc.cluster.local  
/var/cache/showoff # getent hosts argocd-server.argocd.svc  
172.20.238.172    argocd-server.argocd.svc.cluster.local  argocd-  
server.argocd.svc.cluster.local argocd-server.argocd.svc  
/var/cache/showoff # getent hosts argocd-server.argocd  
172.20.238.172    argocd-server.argocd.svc.cluster.local  argocd-  
server.argocd.svc.cluster.local argocd-server.argocd
```

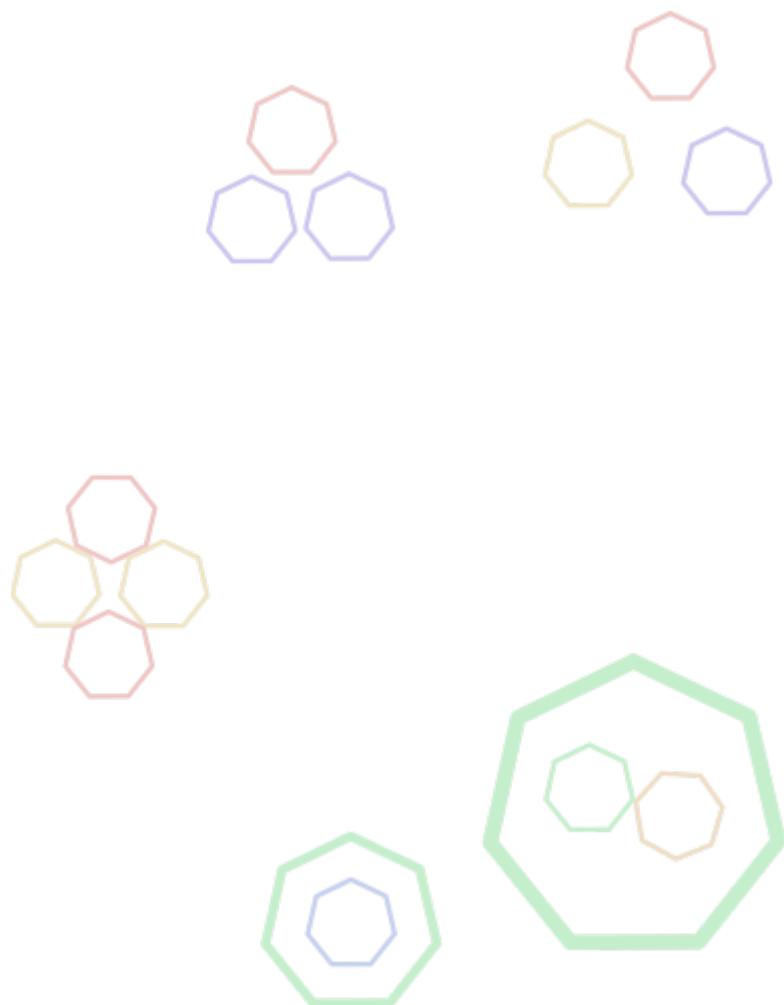


Many new TLDs might clash with namespaces



Types of Services

- Controlled by `spec.type` field
- `ExternalName`: DNS alias to external services
- Cumulative features
 - `ClusterIP`: Internal use, from Pods to Services. This is the default
 - `NodePort`: Publish Service on a "random" (high) port on each host of the cluster
 - `LoadBalancer`: Provision load balancer from cloud provider (IaaS)



Headless Services

- type: ClusterIP + clusterIP: None :

```
kind: Service
apiVersion: v1
metadata:
  name: apache-headless
spec:
  type: ClusterIP
  clusterIP: None
  selector:
    app: apache
```

- no proxy, no load balancing
- DNS: service resolve Pods IP with round robin



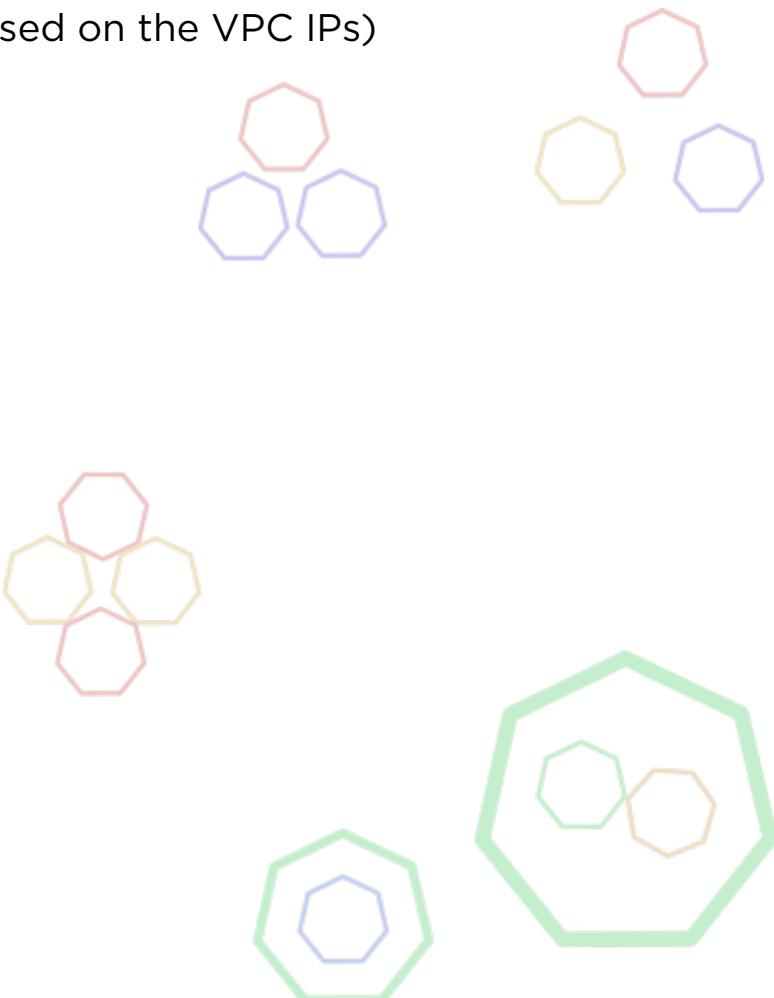
```
# getent hosts apache
172.20.81.5      apache.default.svc.cluster.local

# getent hosts apache-headless
10.50.1.219      apache-headless.default.svc.cluster.local
10.50.1.9        apache-headless.default.svc.cluster.local
```

- Example usage: direct access to Pod (sticky)

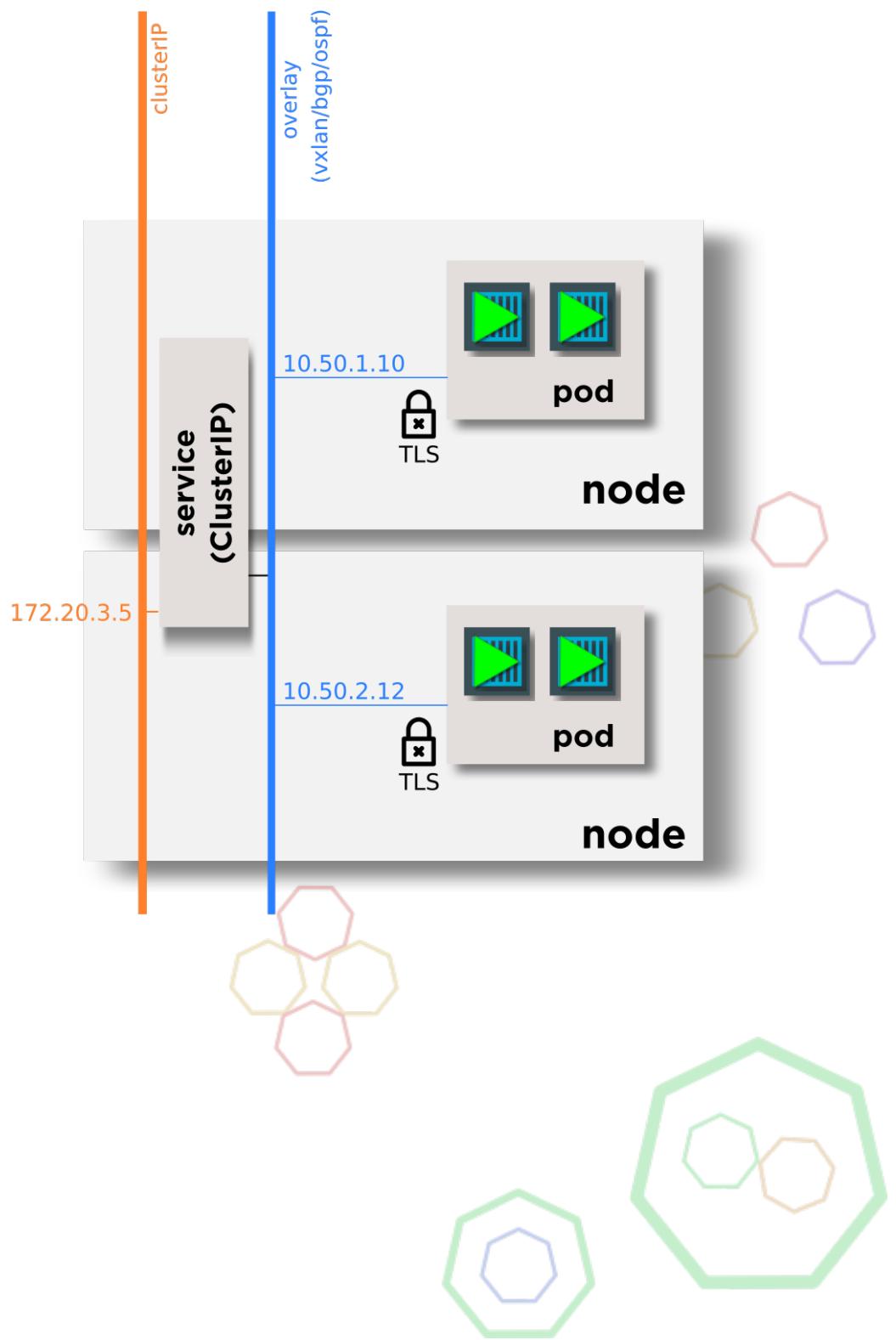
Kubernetes Pod Network: CNI

- CNI (Container Network Interface) specification
- No standard implementation!
- Many implementations:
 - Flannel (VXLAN)
 - Calico (BGP + NetworkPolicies)
 - Canal (Flannel + Calico NetworkPolicies)
 - Cilium (eBPF)
 - Kube Router (BGP)
- OpenShift has its own SDN (Software Defined Network)
- AWS EKS has its own SDN (based on the VPC IPs)



ClusterIP Service: Schema

Let Pods communicate through an internal load-balancer

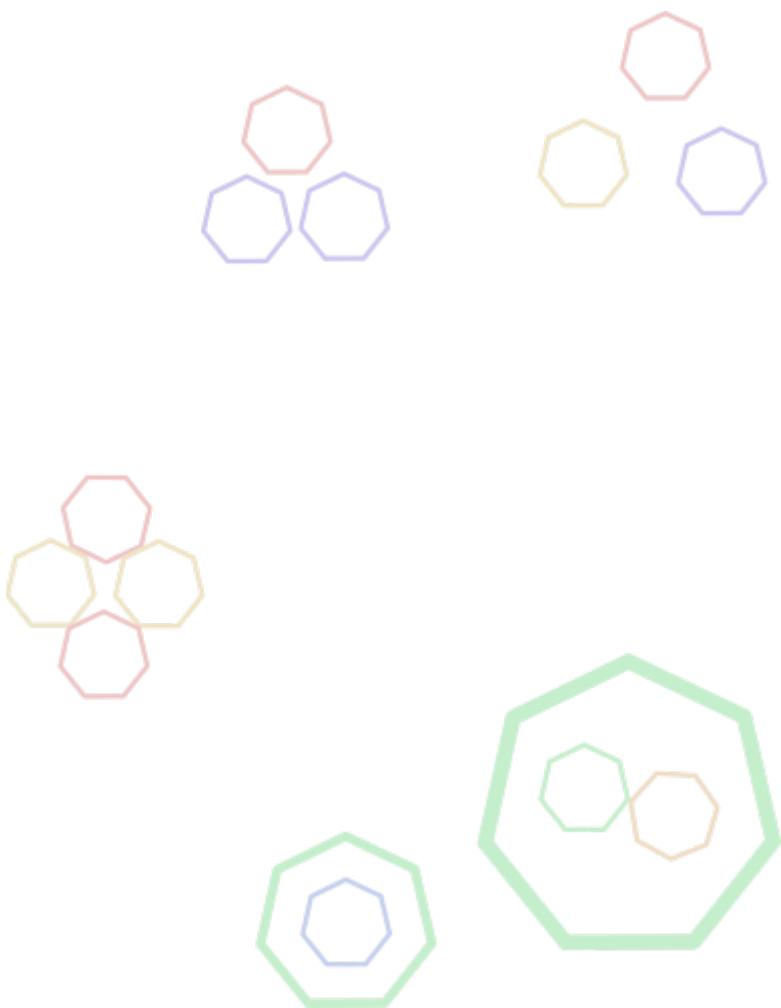


NodePort Service: Principle

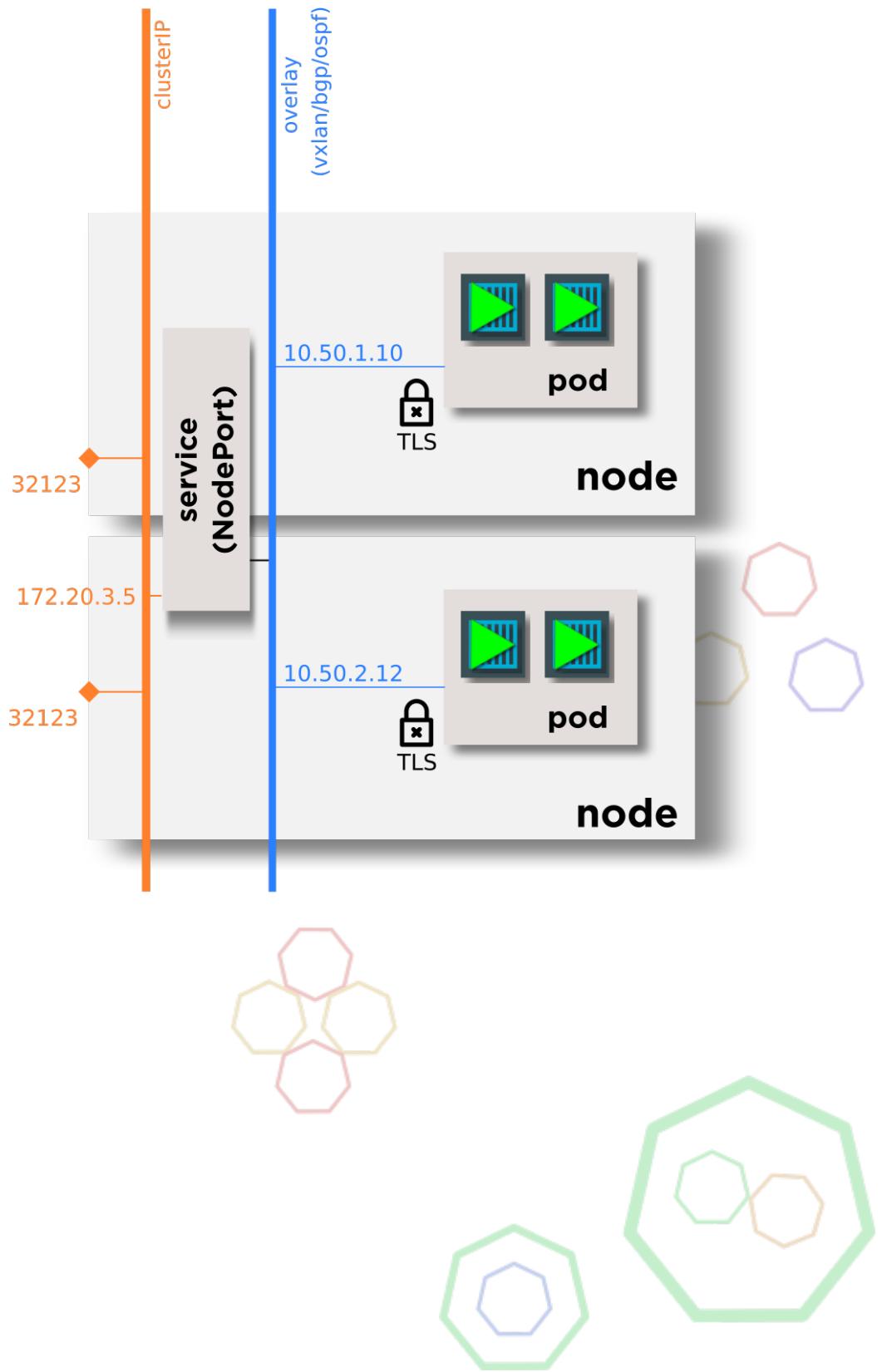


Also creates a `clusterIP`

- Unprivileged port in a configured range. Default is 30000-32767.
- `spec.ports.nodePort` can be used to explicitly set port



NodePort Service: Schema

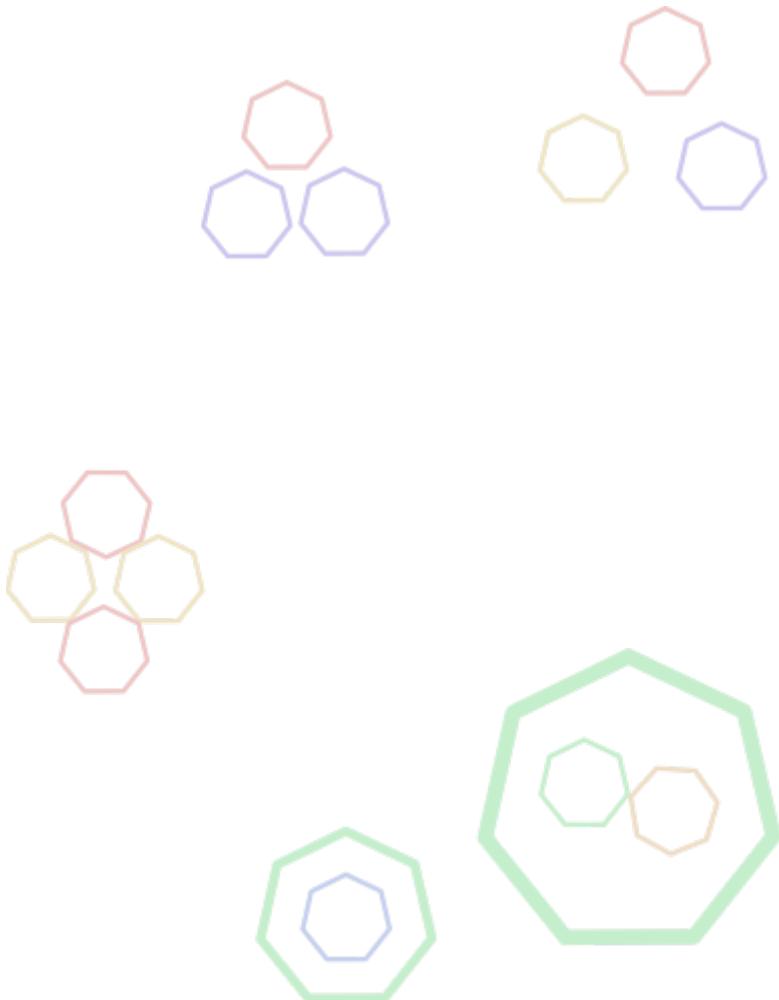


NodePort Service: Syntax

```
kind: Service
apiVersion: v1
metadata:
  name: apache
spec:
  selector:
    app: apache
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
    NodePort: 32000
  type: NodePort
```



If specifying nodePort explicitly, you need to manage port conflicts yourself

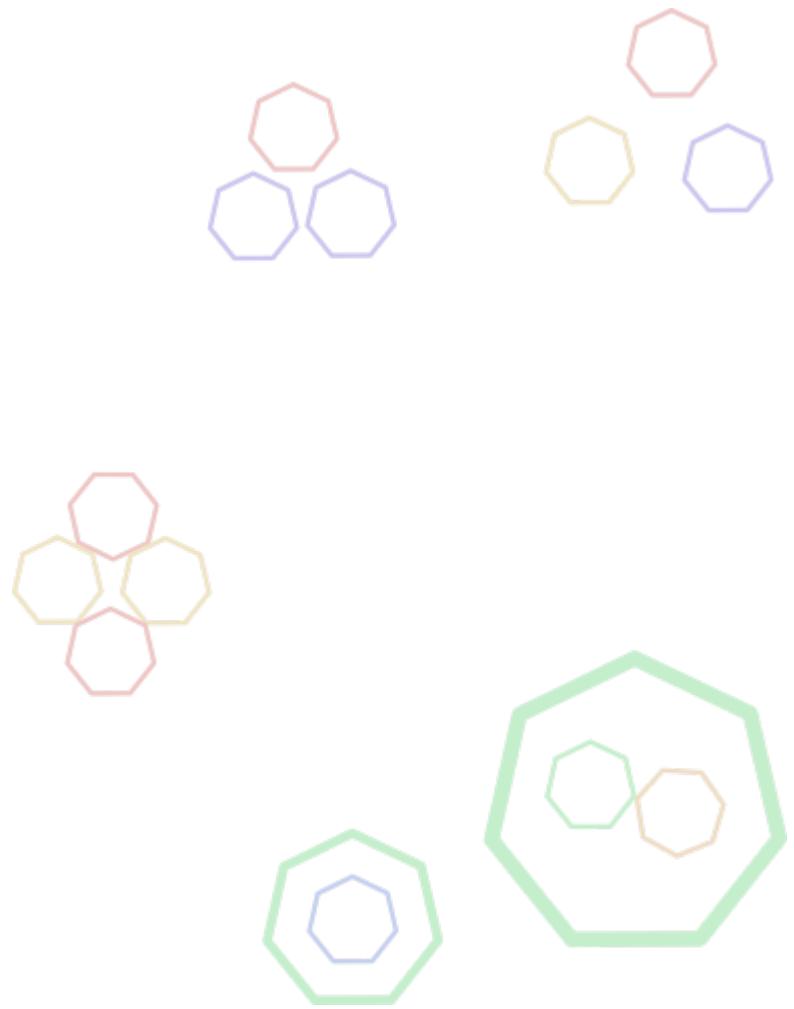


LoadBalancer Service: Principle

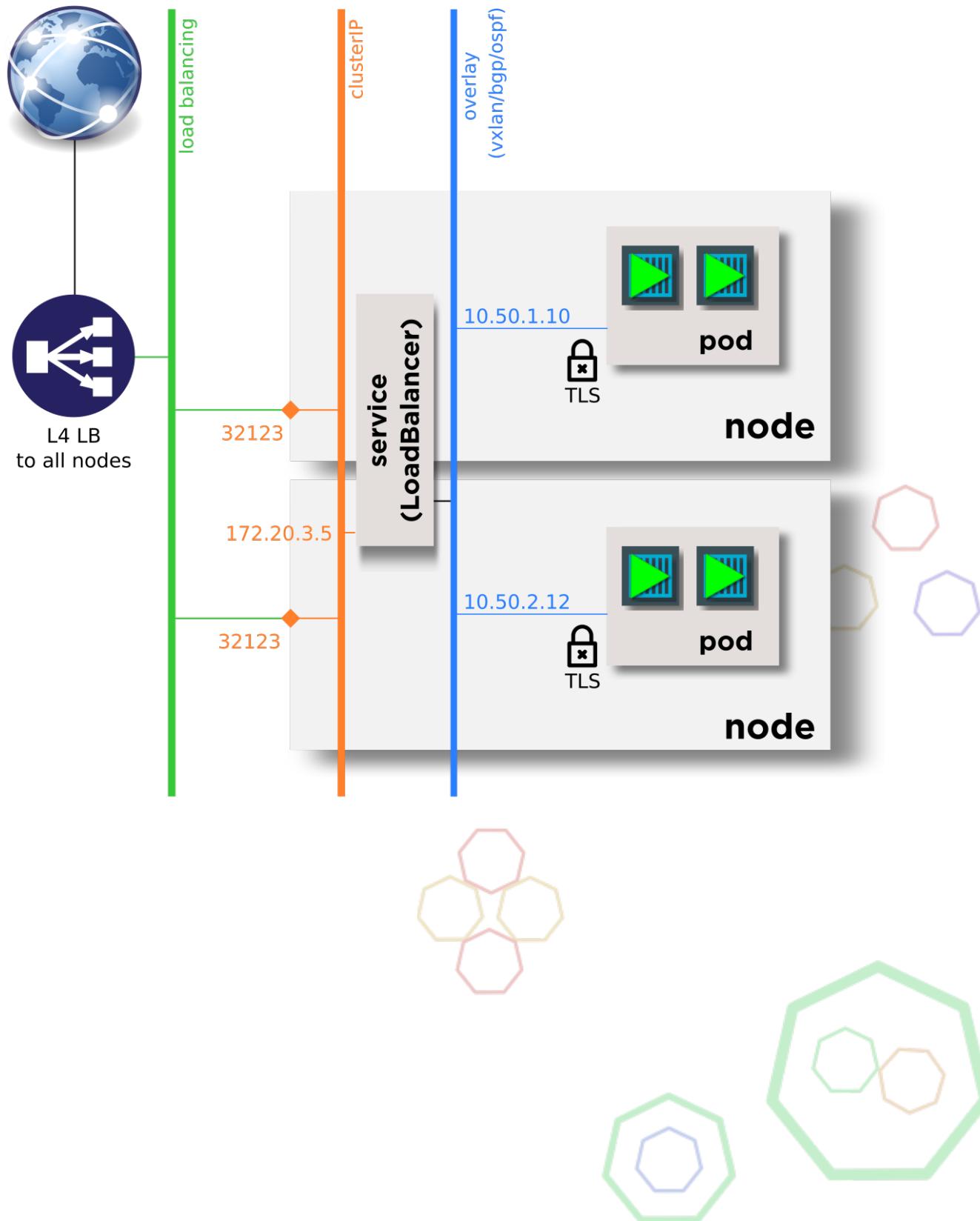


Also creates a `NodePort`

- Provisions a load balancer in the IaaS **Cloud Provider**
- Mainly supported by AWS, GCE, Azure and OpenStack
- Some providers support setting `loadBalancerIP`

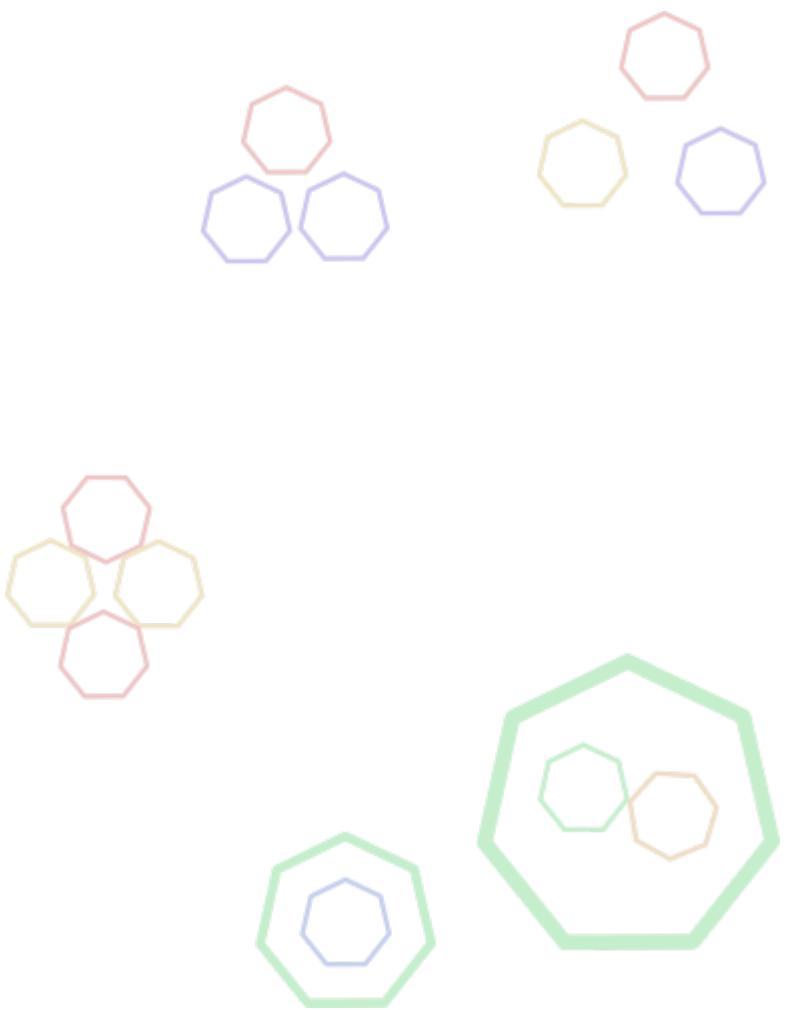


LoadBalancer Service: Schema



LoadBalancer Service: Syntax

```
kind: Service
apiVersion: v1
metadata:
  name: apache
spec:
  selector:
    app: apache
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
  loadBalancerIP: 118.23.42.91
```



ExternalName Service

- no proxy, no load balancing
- DNS: service resolves with a CNAME
- external services, such as databases

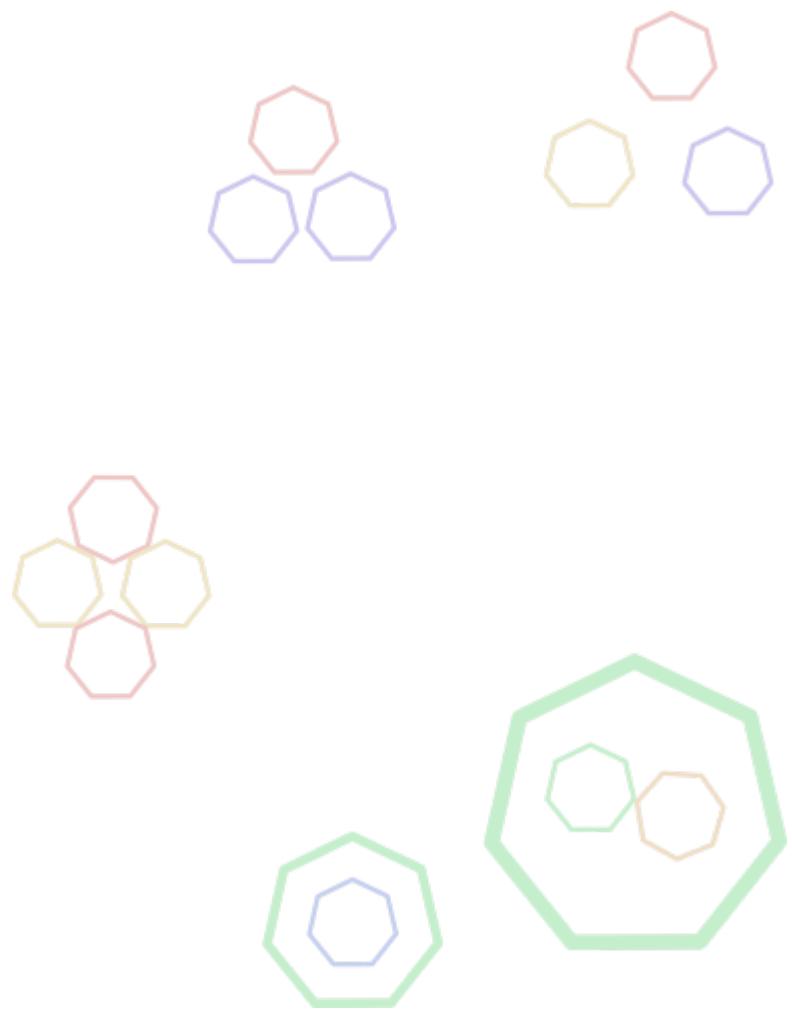
```
kind: Service
apiVersion: v1
metadata:
  name: database
spec:
  type: ExternalName
  externalName: my.database.example.com
```



Mapping Services to External IPs

- Use an `ExternalName` service with a `nip.io` or `xip.io` entry
- Create a Service with an empty selector: `selector: {}`
- Create Endpoints:

```
kind: Endpoints
apiVersion: v1
metadata:
  name: database
subsets:
- addresses:
  - ip: 192.168.1.45
ports:
- port: 5432
  name: postgres
```



Lab 23.1: Create Services



Objective

Deploy frontend and connect to backend

Steps

- Create a Service for backend
- Create a Deployment for the frontend using `ghcr.io/camptocamp/course_docker_frontend:latest`
- Configure frontend to access backend using :
 - `BACKEND_HOST`
 - `BACKEND_PORT`
- Create a Service for frontend

Questions

- Can we create a Service that abstracts both the backend and frontend Pods?
 - yes
 - no



Lab 23.2: Install Database and Access it

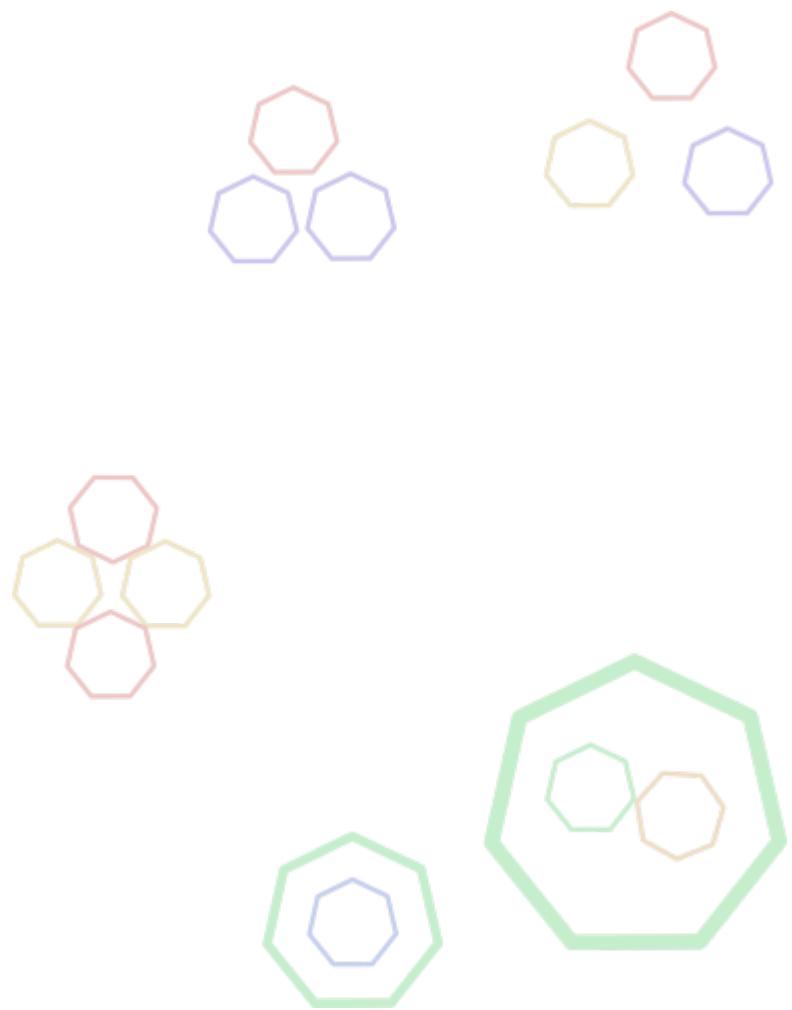


Objective

- Create an ephemeral database and a Service to access it

Steps

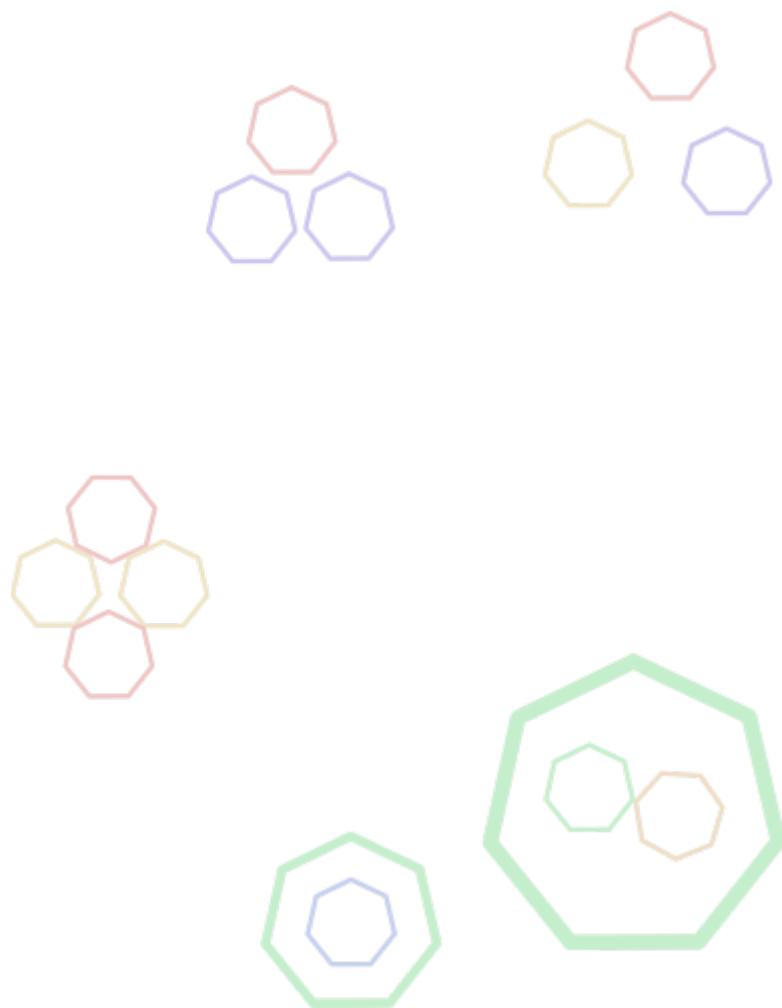
- Create a Deployment for a PostgreSQL database
- Add Liveness and Readiness checks (TCP)
- Create a LoadBalancer Service to access the database from outside the cluster
- Check access from workstation



Checkpoint: Services

- Which types of Services can we use to access a database outside the cluster?
 - ClusterIP
 - NodePort
 - LoadBalancer
 - ExternalName

- Which types of Services can we use if the external database only has an IP address?
 - ClusterIP
 - NodePort
 - LoadBalancer
 - ExternalName



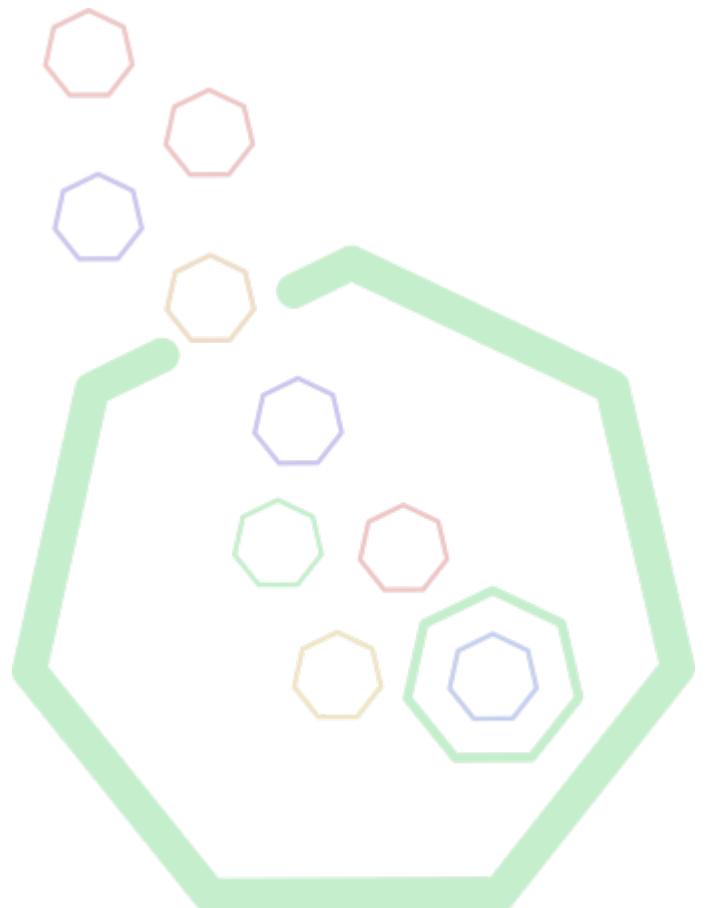
INGRESSES

Lesson 24: Ingress

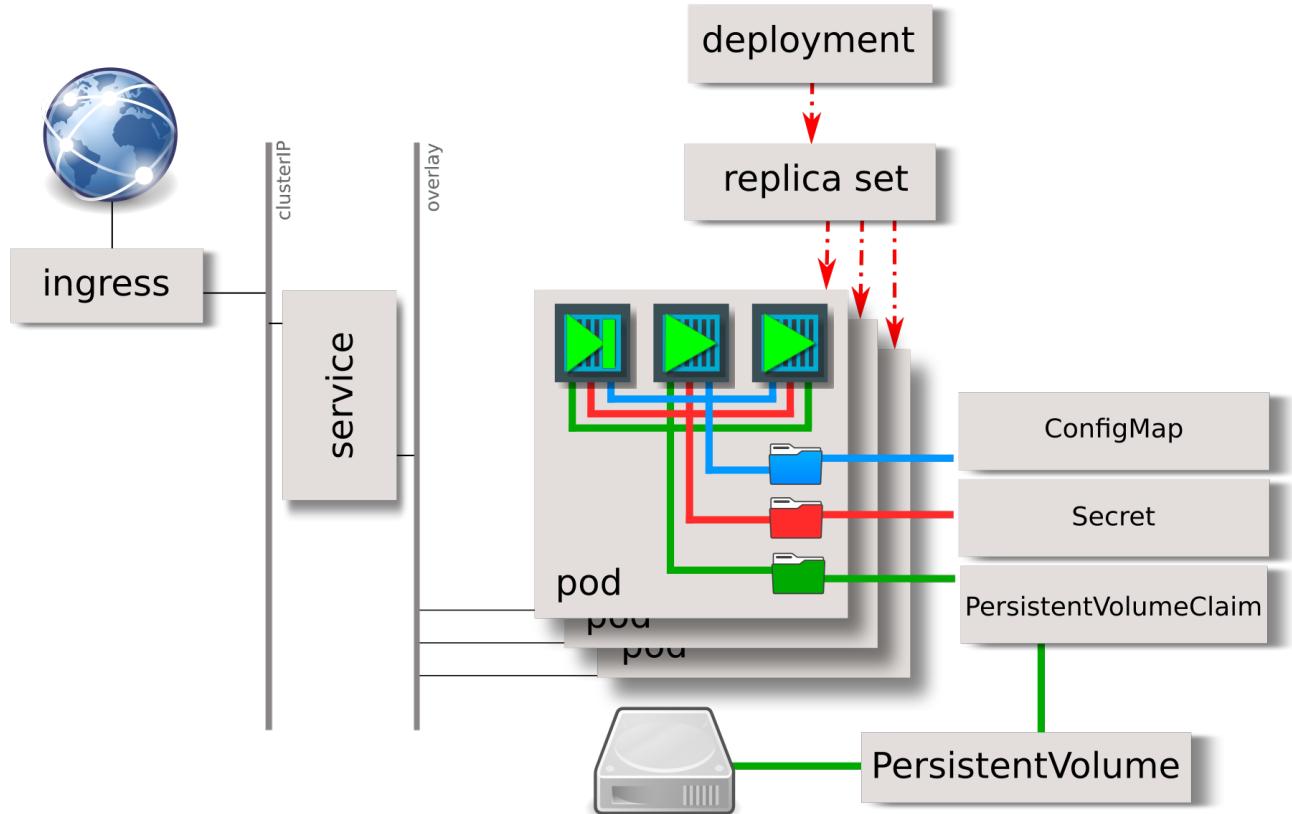
Objectives

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

- Route HTTP traffic to multiple backends
- Understand how Ingress objects interact with IaaS
- Expose services on HTTPS with an SSL-terminated Load Balancer

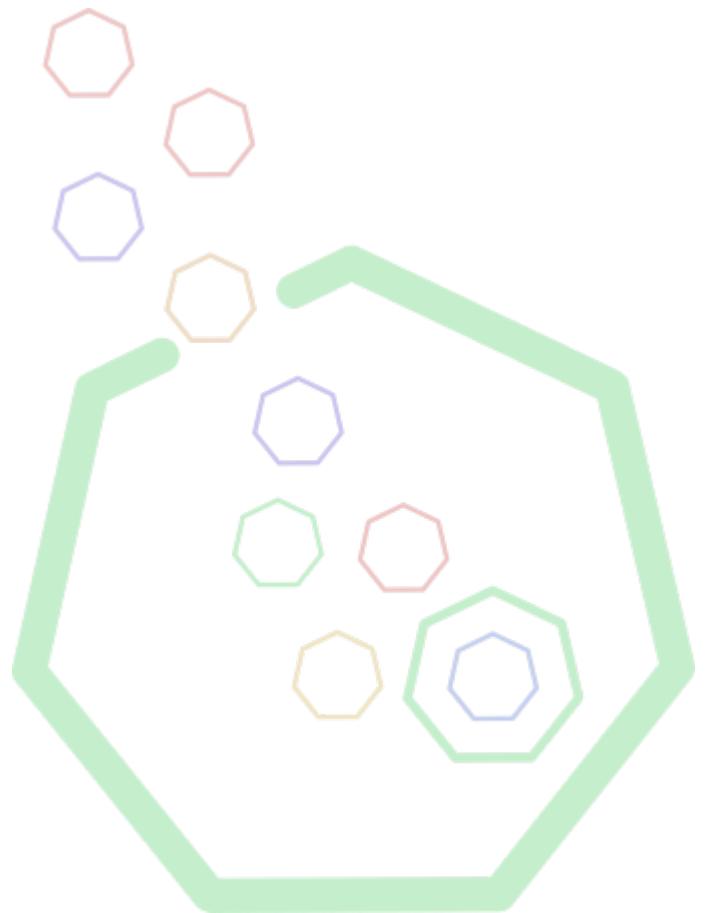


API Objects Overview

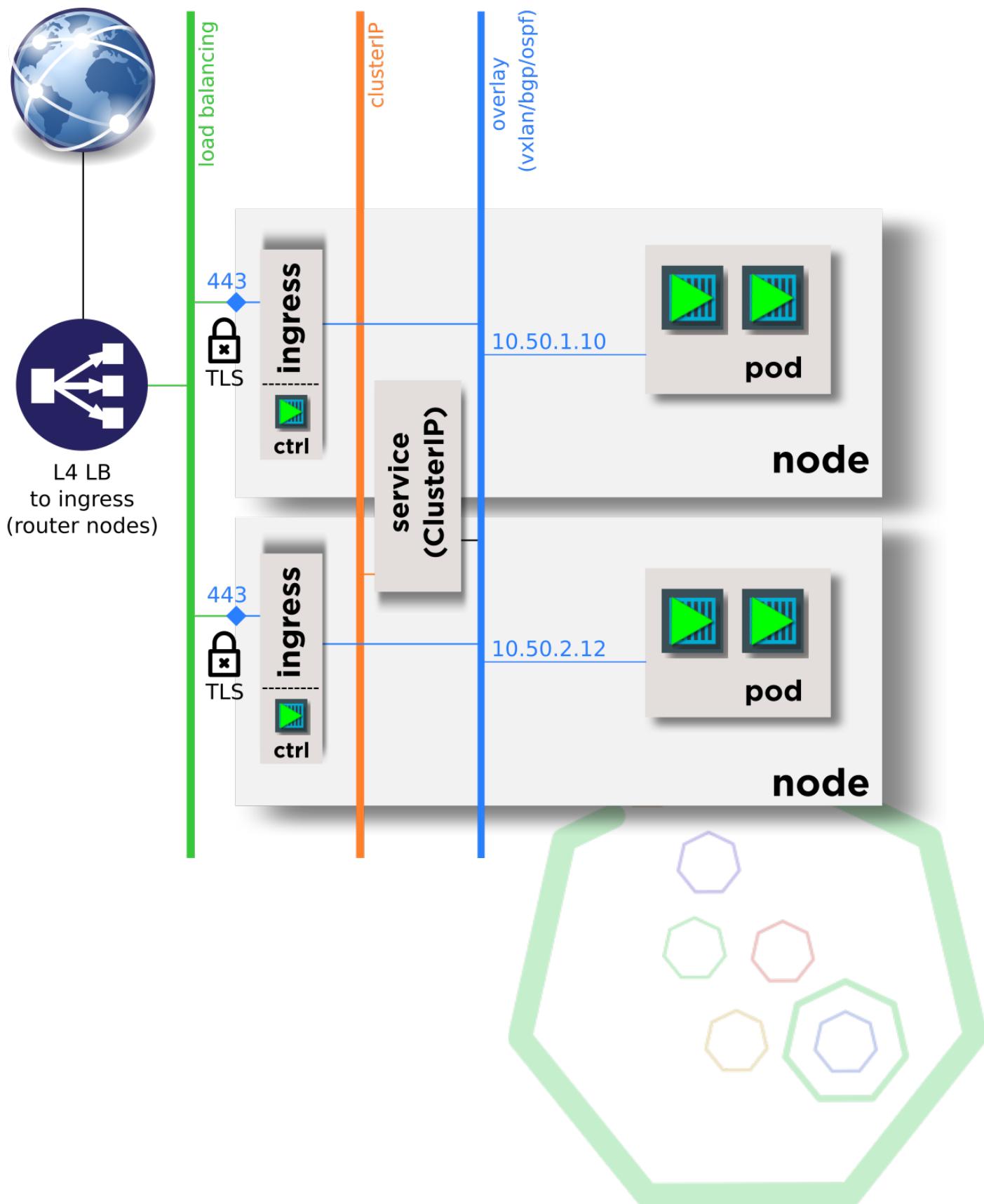


Ingress

- Connect the external world to Services with a Layer 7 proxy
- Load balancing
- SSL termination (optional)
- Name-based virtual hosting



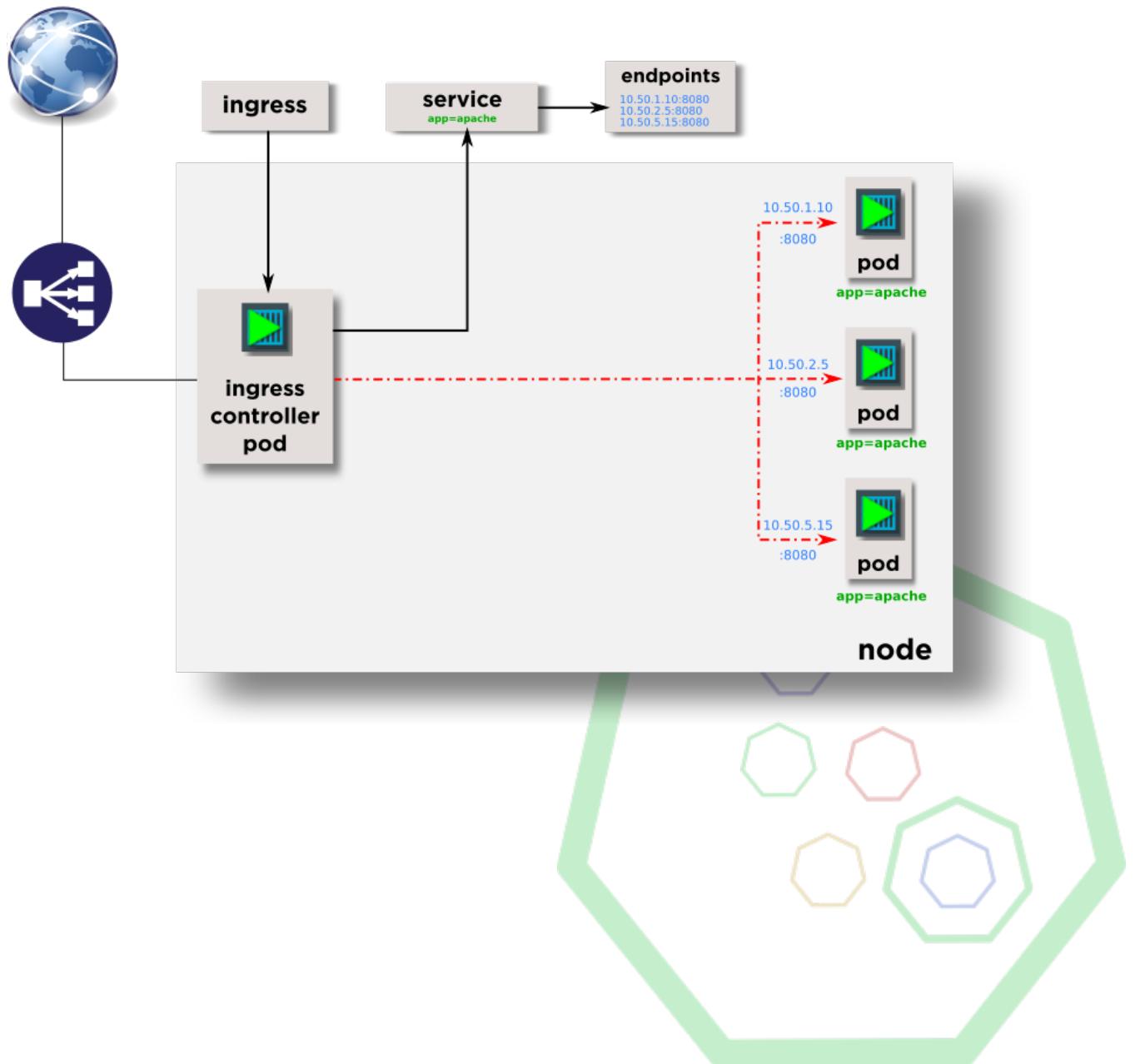
Ingress Controller: Principle



Ingress Controller: Implementation

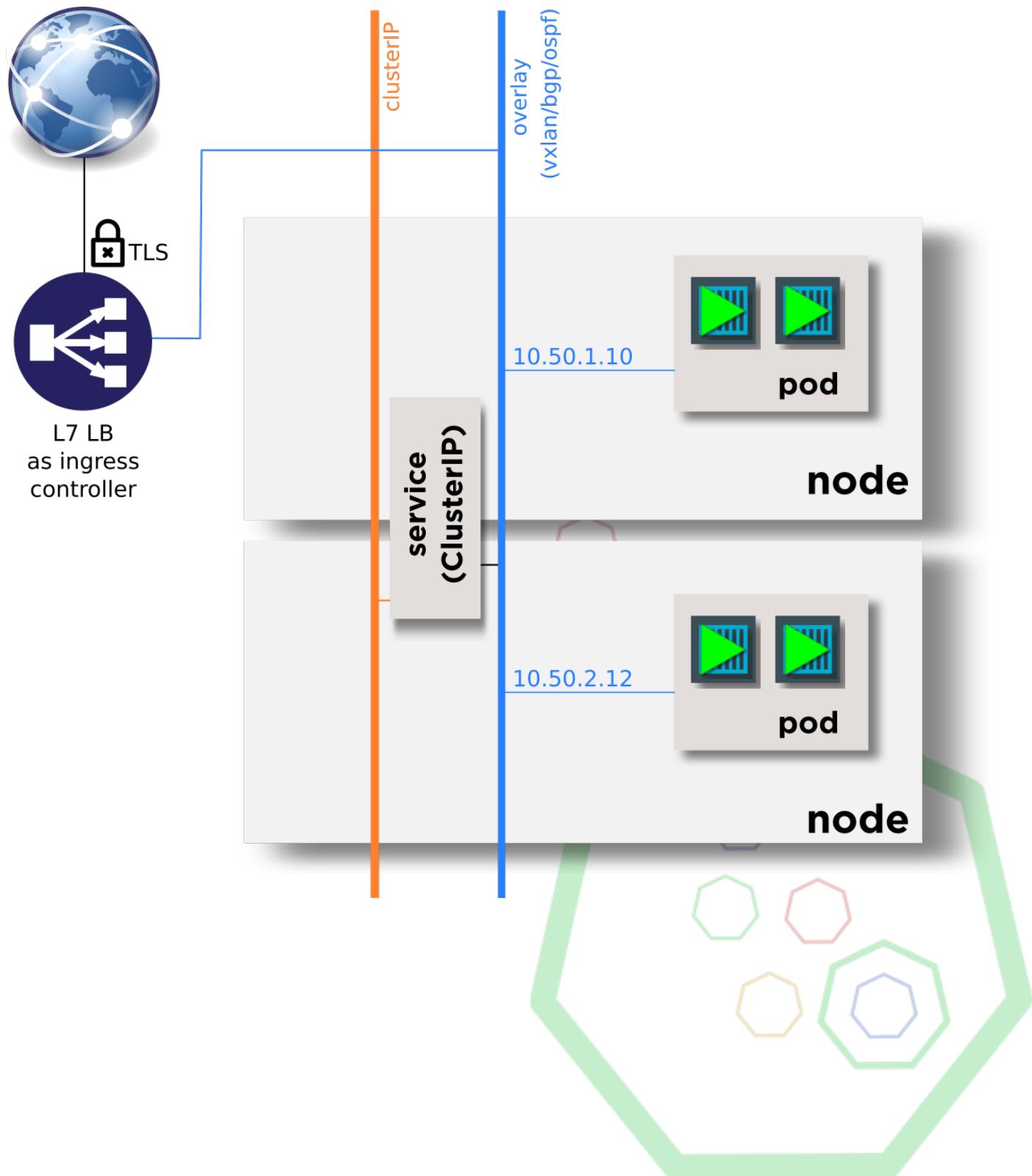
i No standard Controller provided

- Users create Ingress Objects (in the Kubernetes API)
- Ingress Controllers implement Ingress creation (e.g. GCE, nginx, Traefik, HAProxy, Contour)



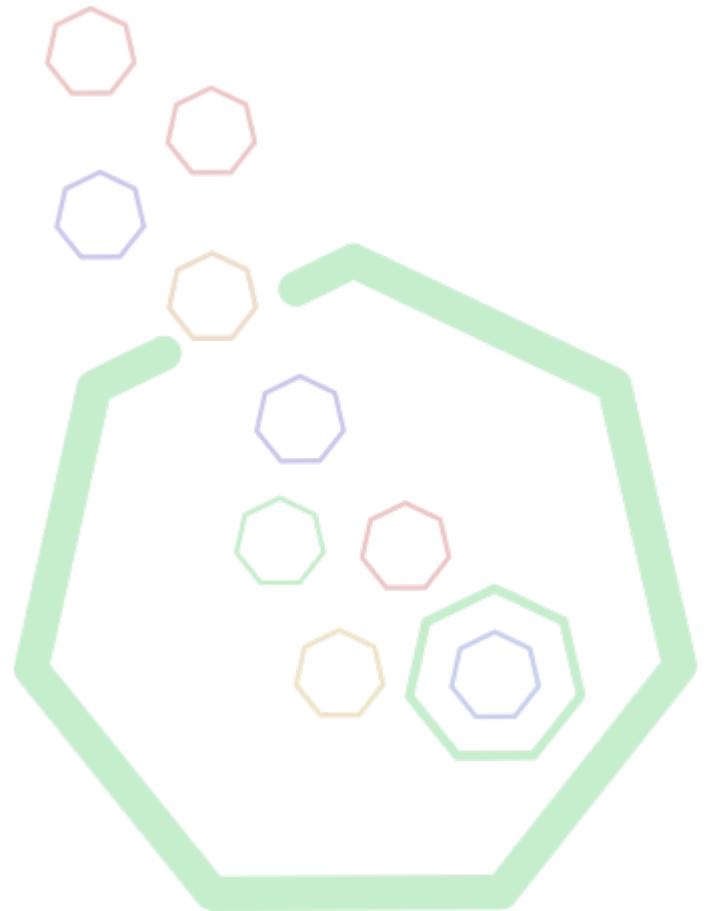
Ingress Controller: External LoadBalancer

Some load balancers (e.g. AWS ALB, F5) support meshing with the overlay network directly



Traefik Ingress Controller

- Runs in Kubernetes as *Pod* (through *Deployment*)
- uses a *Service* `spec.type = NodePort` to bind a port 80 and 443
- Controller registers API callbacks and maintains Traefik configuration
- Work queue to aggregate changes: severals changes in short time frame only generate one update.

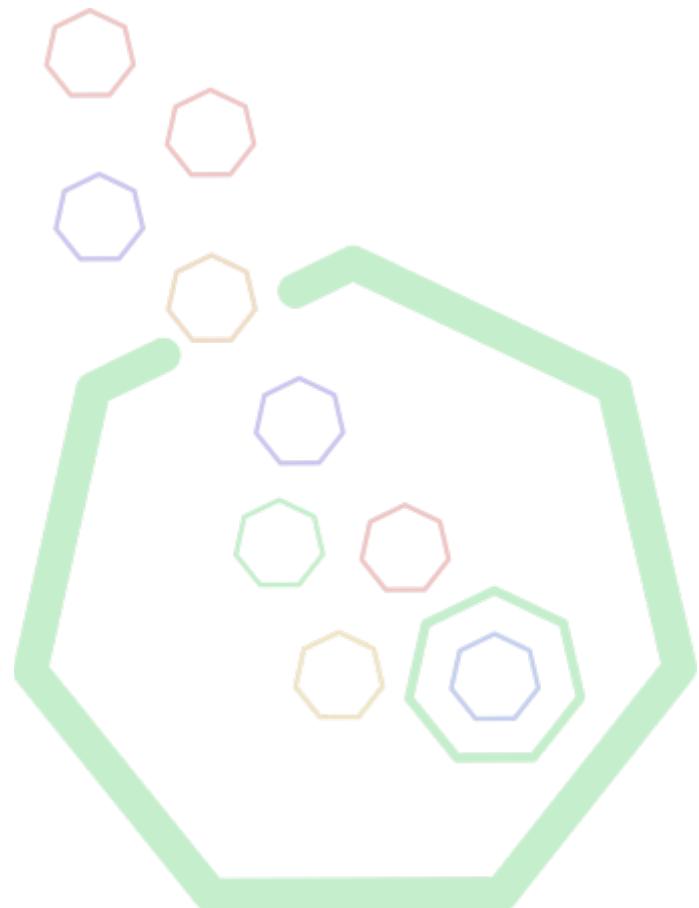


Traefik Ingress Deployment and Usage



Instructor demonstrates:

- Deployment of the Traefik Ingress on the Kubernetes cluster
- How to use an Ingress object on the cluster



Ingress Syntax

- Specify an hostname (virtualhost)
- Define a mapping between paths and services

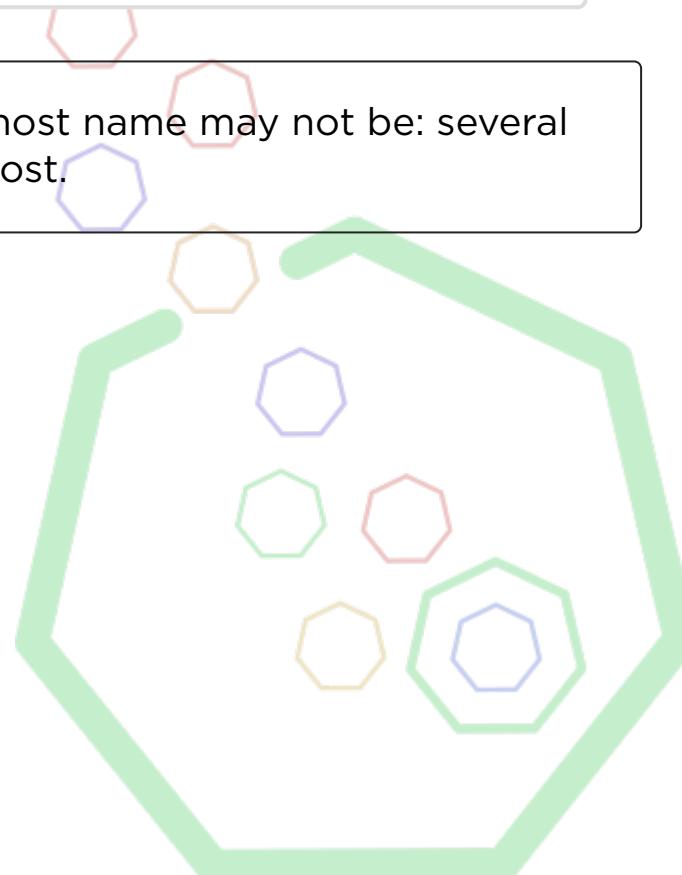
Ingress controllers allow additional configurations with annotations or **custom object**:

- Rewrite path, manage headers
- Configure redirection

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: apache
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: apache
          servicePort: 80
```



Ingress name must be unique, host name may not be: several Ingress objects can use same host.



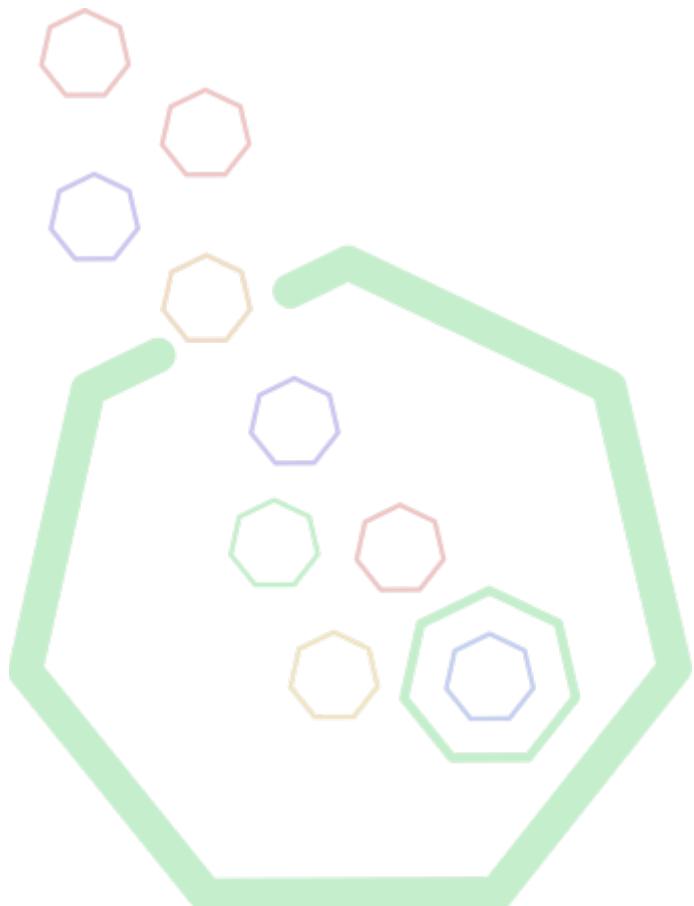
Ingress Syntax v1

Version `networking.k8s.io/v1` introduced in Kubernetes 1.19:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: apache
spec:
  rules:
  - http:
    paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: test
          port:
            number: 80
```



The `extensions/v1beta1` and `networking.k8s.io/v1beta1` API versions of Ingress will no longer be served in v1.22.



HTTPS Load Balancing

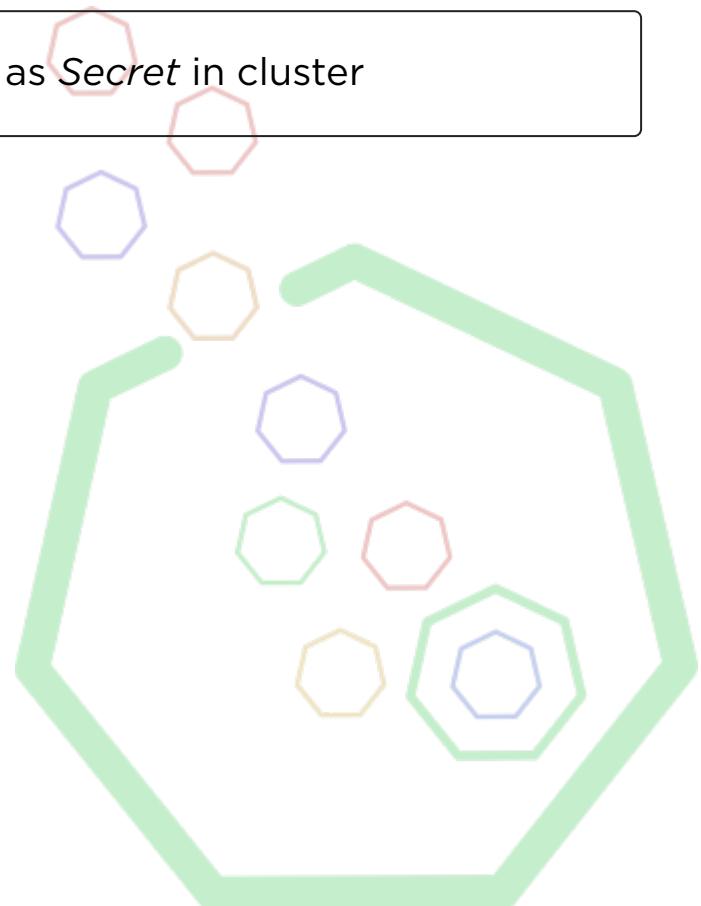
Add a `spec.tls` section:

- List of hostnames
- Name of the Secret containing the certificate and key

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: apache
spec:
  tls:
    - hosts:
        - my-app.example.com
      secretName: my-certificate
  rules:
    - host: my-app.example.com
      http:
        paths:
          - path: /
            backend:
              serviceName: apache
              servicePort: 80
```



Requires an existing certificate as *Secret* in cluster



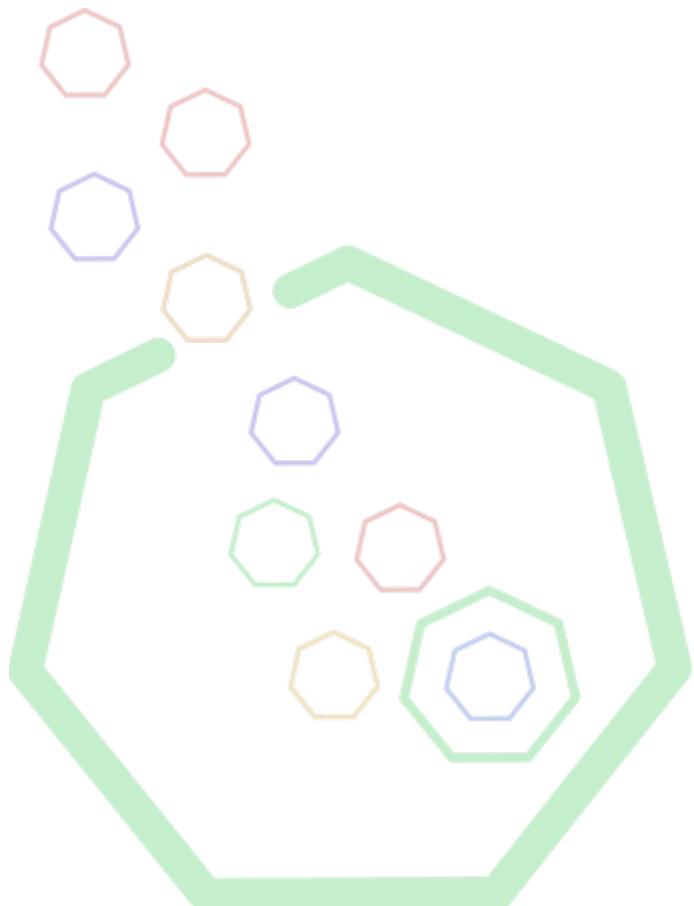
Default Backend

- Requests that don't match host and/or path are redirected to the *default backend*
- Can be used to implement 404 pages for example.
- The *default backend* is an Ingress with no `rules` section:

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: default-ingress
spec:
  backend:
    serviceName: default-webserver
    servicePort: 80
```



You still need to create a Service and the workload behind



Lab 24.1: Create Ingress



Objective

- Create an Ingress to access the consumer service from outside

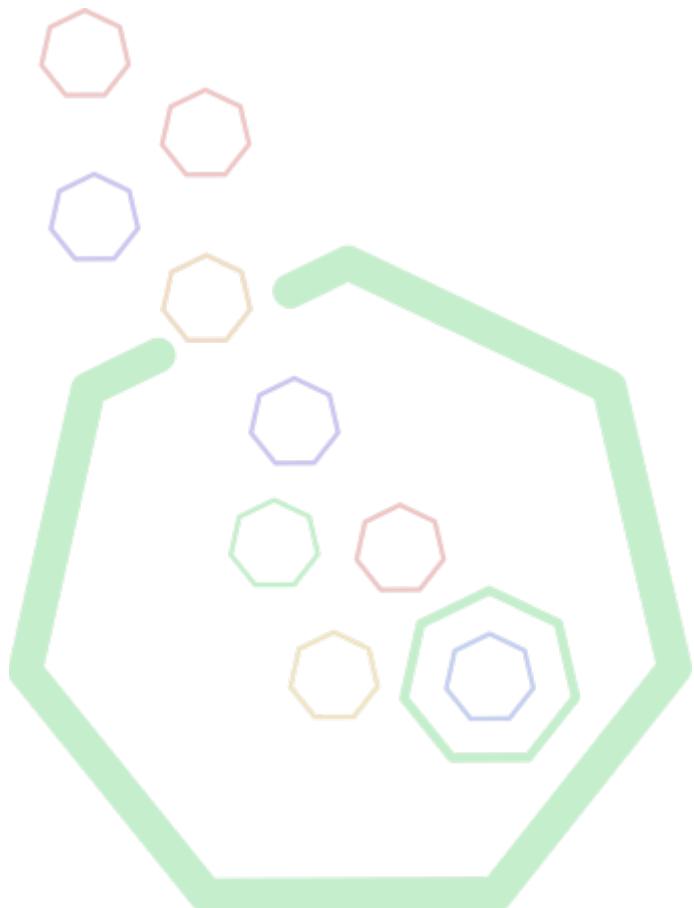
Steps

- Create a Ingress object
- Inspect the created object to check results

Checkpoint: Ingress

- Can we create more than one Ingress with the same host?
 - Yes
 - No

- Is it possible to perform Layer 4 load balancing with Ingress?
 - Yes
 - No



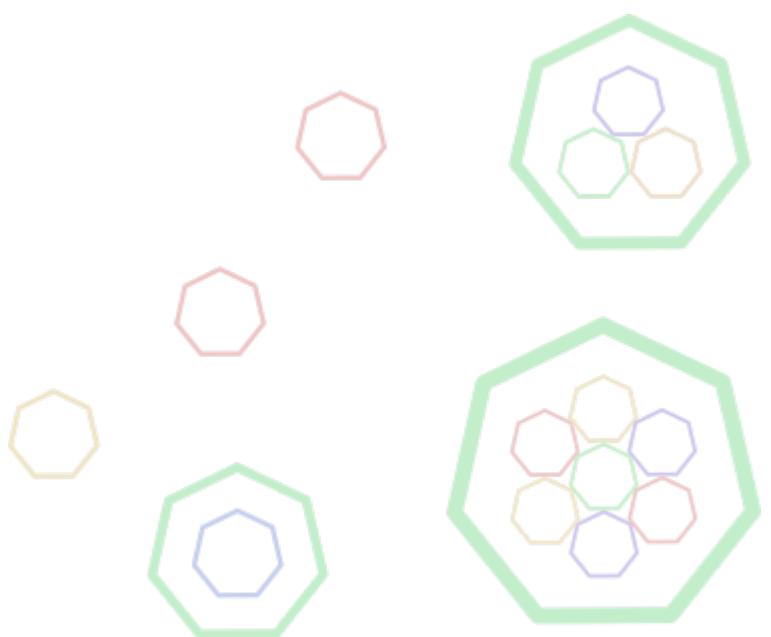
DATA MANAGEMENT

Lesson 25: Data Management

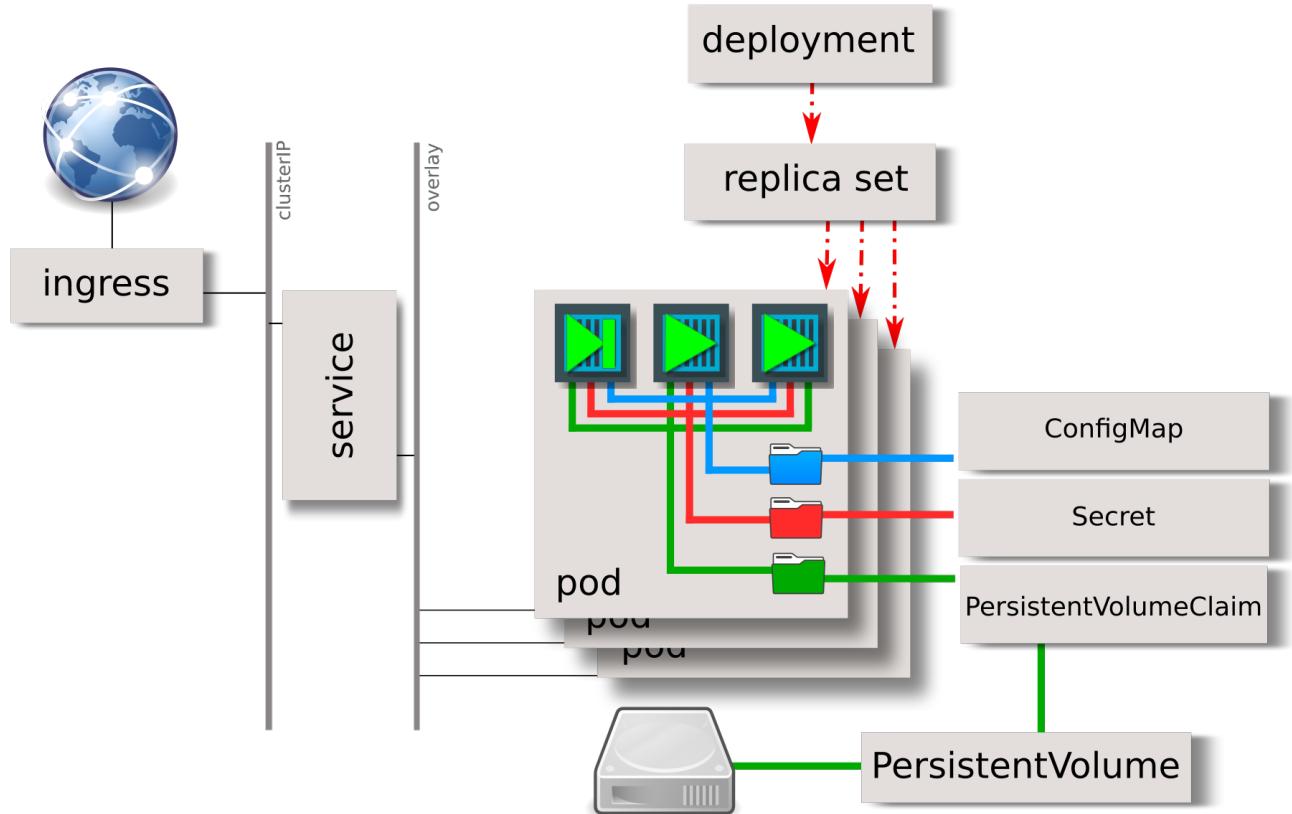
Objectives

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

- Use persistent storage
- Manage volumes
- Understand Storage Class dynamic provisioning

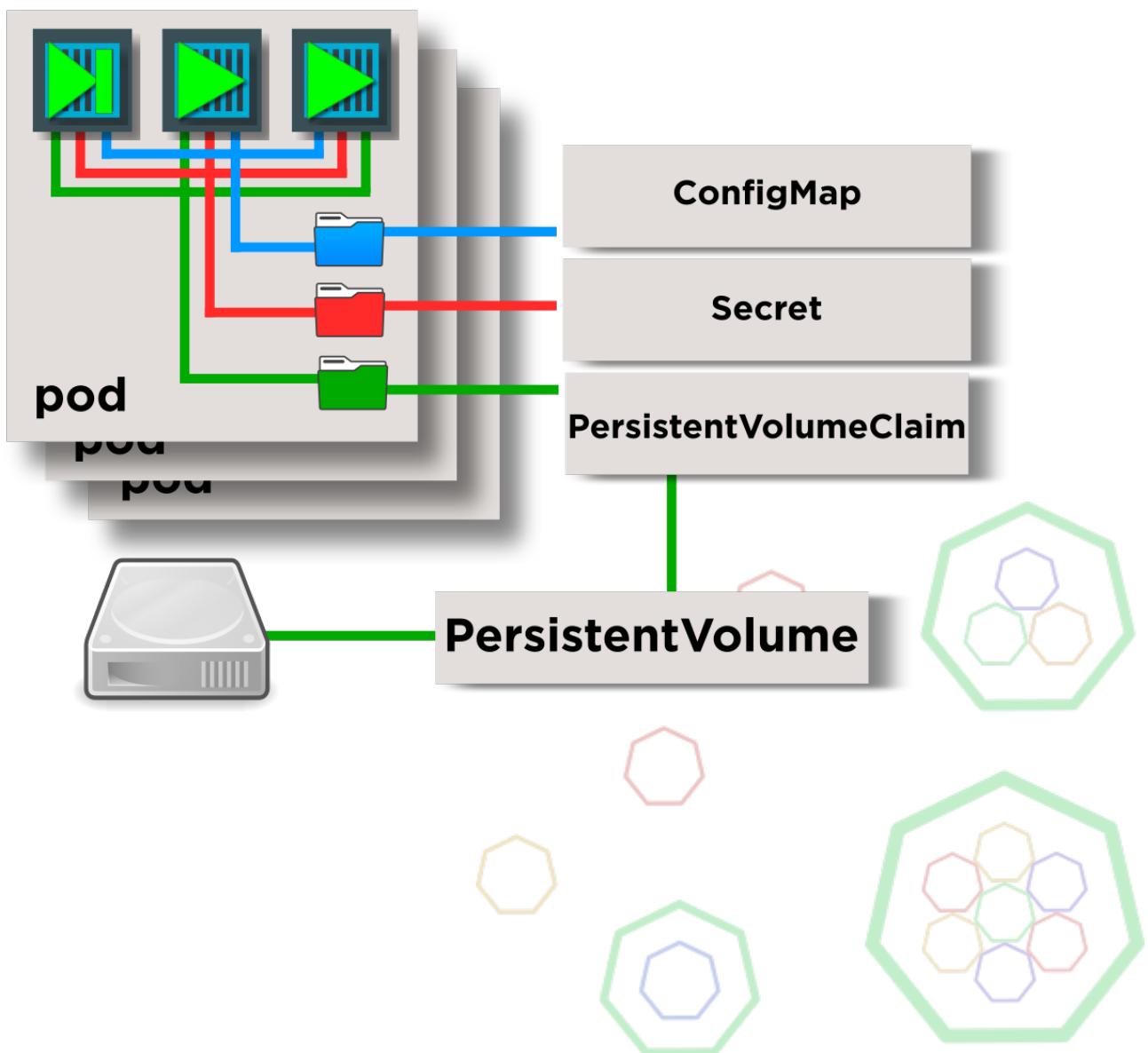


API Objects Overview



Comparison with Docker Volumes

- Like in Docker, writes to container FS are not persisted
- Containers in Pods share data/volumes:
 - Bootstrap/Config containers need to write config to other containers
 - Some folders/files need to be kept across container reboots

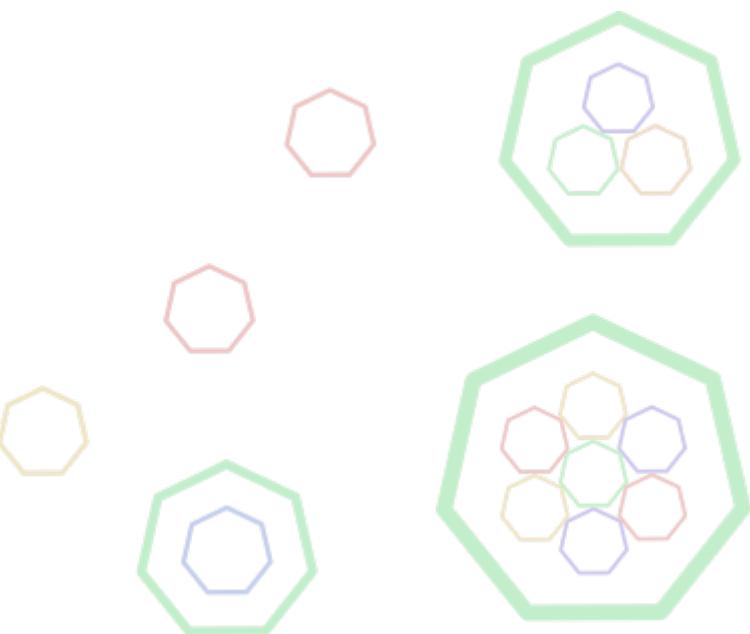


Ephemeral Volumes

- No persistence
- Used for
 - cache (`emptyDir`)
 - configuration (`ConfigMap`)
 - secrets (`Secret`)
 - shared volumes (`emptyDir`, inside `Pods`)

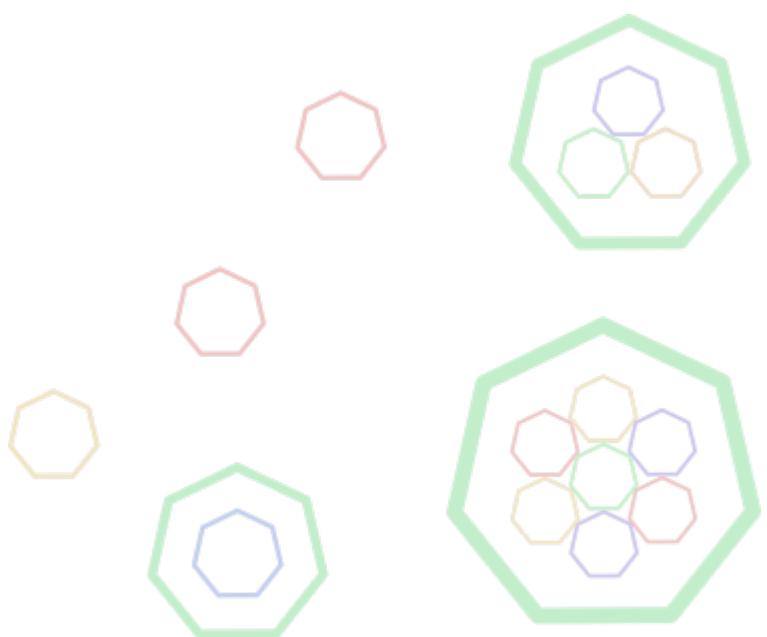
Example with `emptyDir`:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```



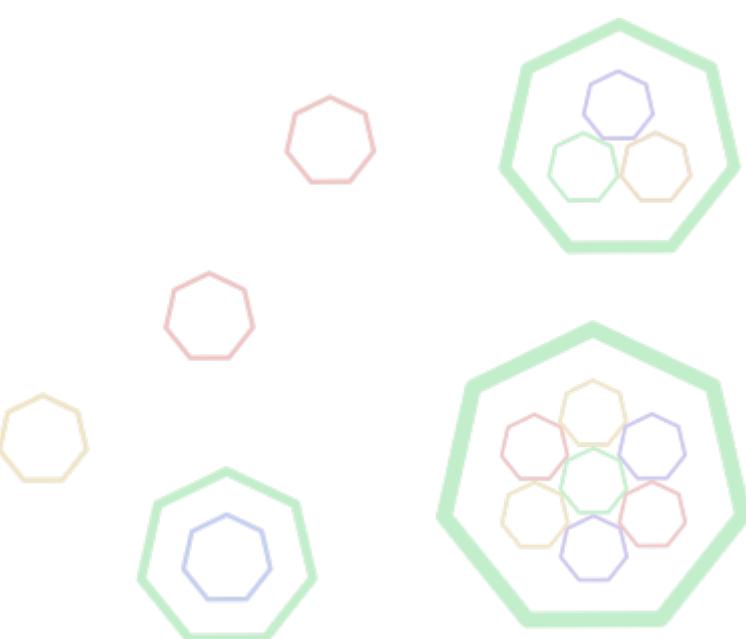
Volume Lifecycle

- Docker has limited management of volumes:
 - Automatically created at first used
 - No lifecycle
- Kubernetes adds a real volume lifecycle
 - Volumes are explicit objects
 - Create, delete volumes like other objects



Volume Drivers

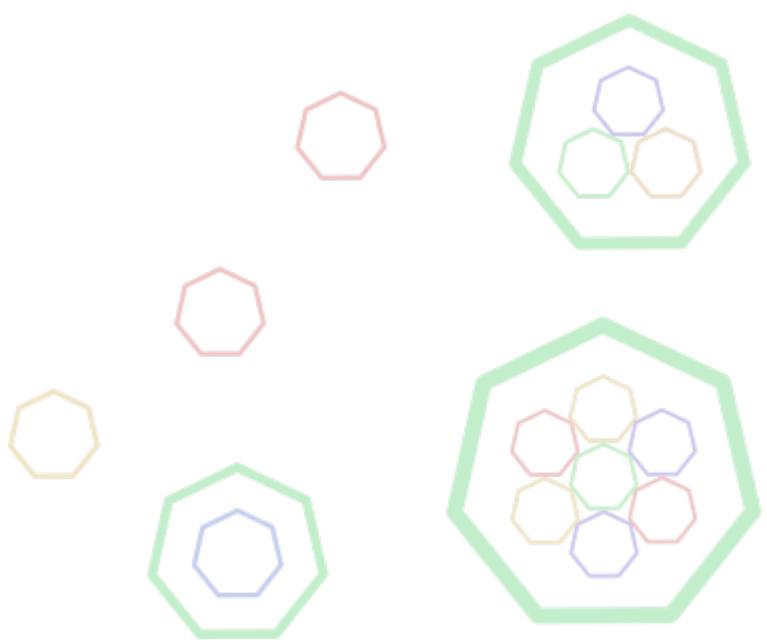
- Volume drivers in Docker are limited:
 - Only one driver per container (cannot mix local and network volume)
- Kubernetes allows many drivers across cluster and Pods
- Example drivers:
 - `emptyDir`, `hostPath`
 - `gcePersistentDisk`, `awsElasticBlockStore`, `azureDisk`
 - `cephfs`, `glusterfs`, `iscsi`, `nfs`
 - `secret`, `configMap`



Using Volumes in Pods

- At Pod level, add a `volumes` section to declare volumes used in Pod
- Volume names are shared between all containers in a Pod
- At Container level, mount the volumes on local paths

```
kind: Pod
apiVersion: v1
metadata:
  name: apache
spec:
  containers:
    - name: apache
      image: httpd:2.4
      volumeMounts:
        - name: website
          mountPath: /usr/local/apache2/htdocs/
  volumes:
    - name: website
      gcePersistentDisk:
        pdName: website-prod
        fsType: ext4
```

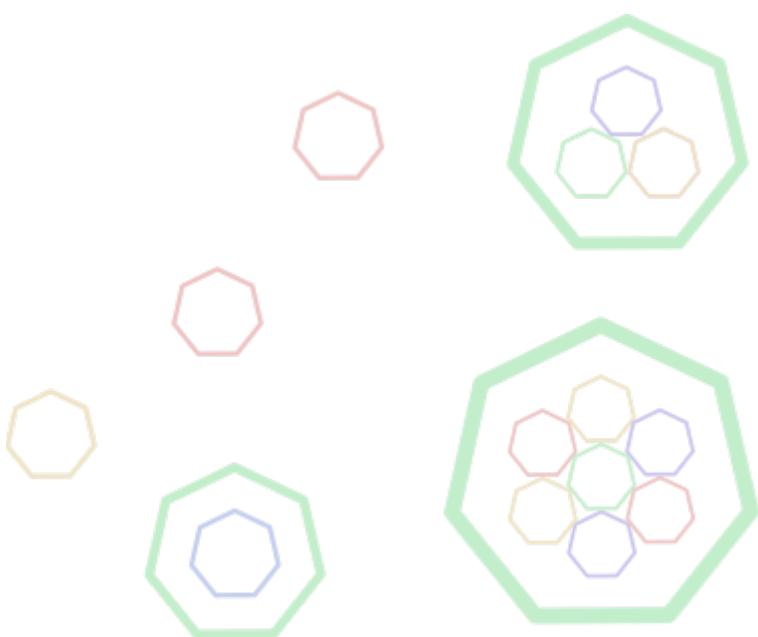


Using Volume subPath

- Using subPath, you can mount a subdirectory of a volume inside a container
- The volume can contain multiple datas:
 - DB storage
 - Source code
 - Static HTML files
 - etc.



Using subPath disables volume updates (ConfigMap, Secret)

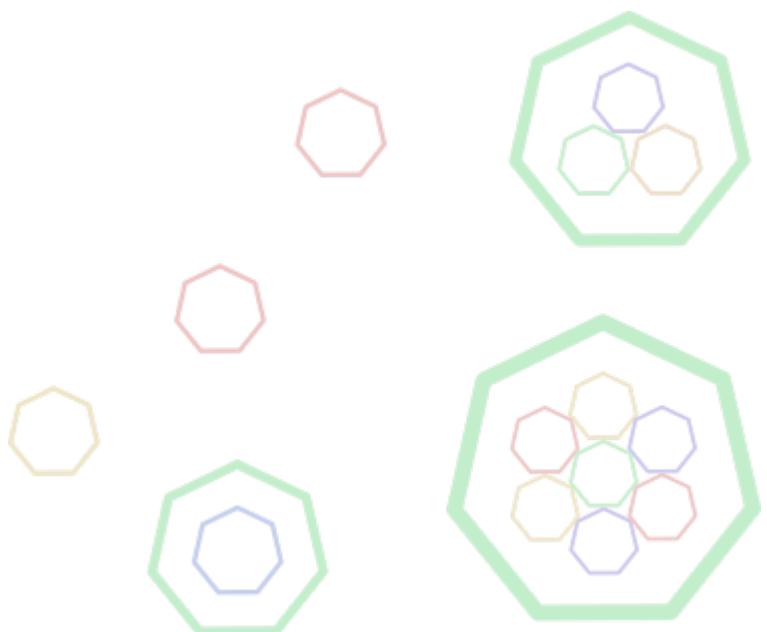


subPath – Example

```
kind: Pod
apiVersion: v1
metadata:
  name: macro-service
spec:
  containers:
    - name: postgres
      image: postgres
      env:
        - name: POSTGRES_PASSWORD
          value: "pgpass"
      volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: data
          subPath: postgres
    - name: php
      image: httpd:2.4
      volumeMounts:
        - mountPath: /var/www/html
          name: data
          subPath: html
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: macro-service-data
```

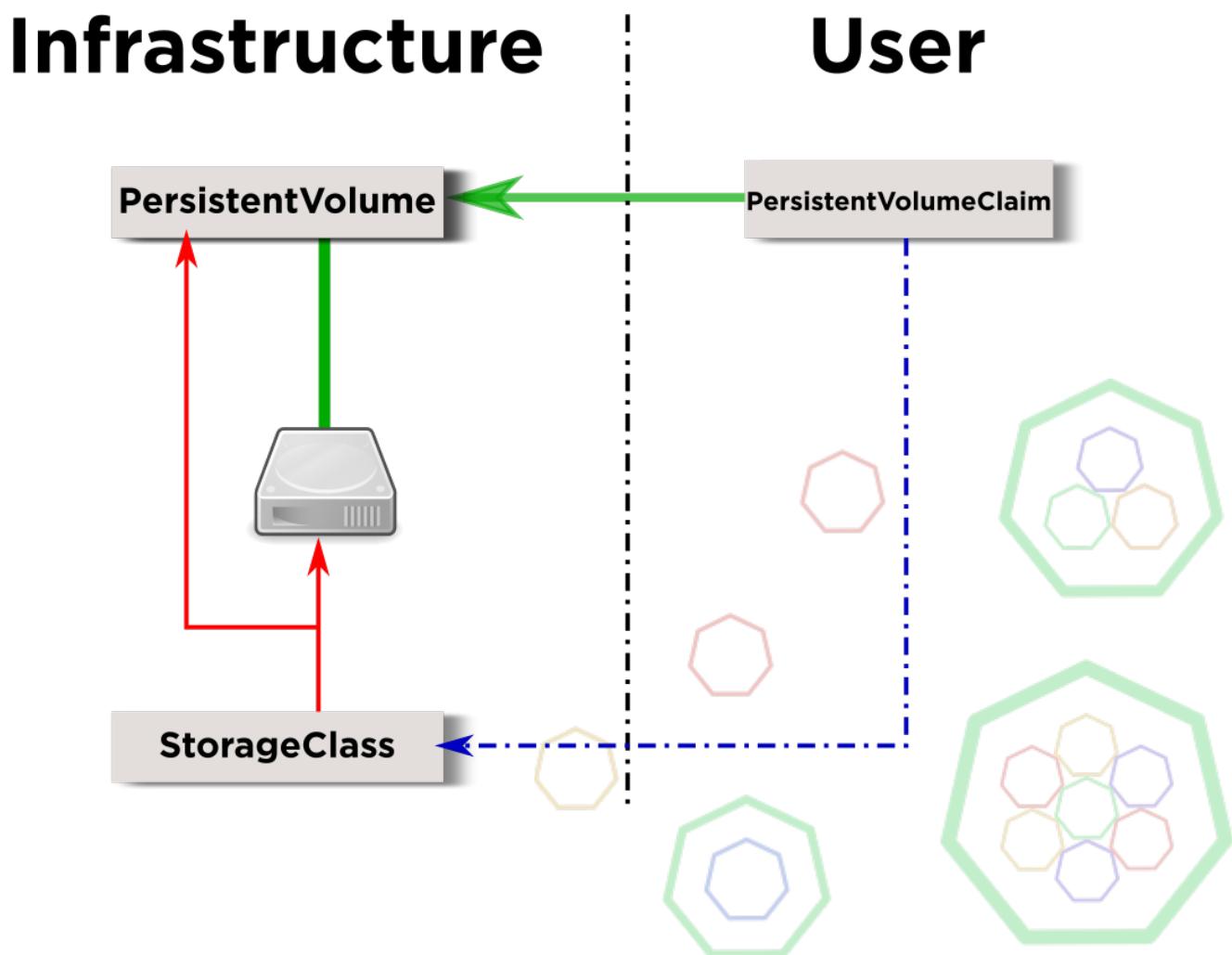


This is a rather ugly example, but it can be useful for integration tests for example



PersistentVolumeClaim — Principle

- Decouple provisionning with usage
- Ops handle provisioning of PersistentVolumes
- Devs claim volumes for usage
- Volume allocation is divided in two parts:
 - PersistentVolumeClaim
 - PersistentVolume
- one-to-one mapping between PVC and PV

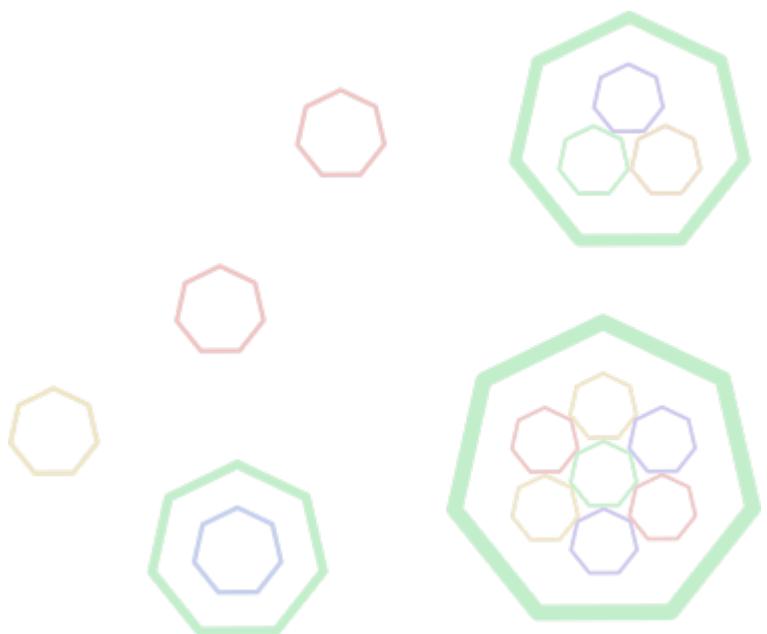


PersistentVolumeClaim – Syntax

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: website-data
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
```

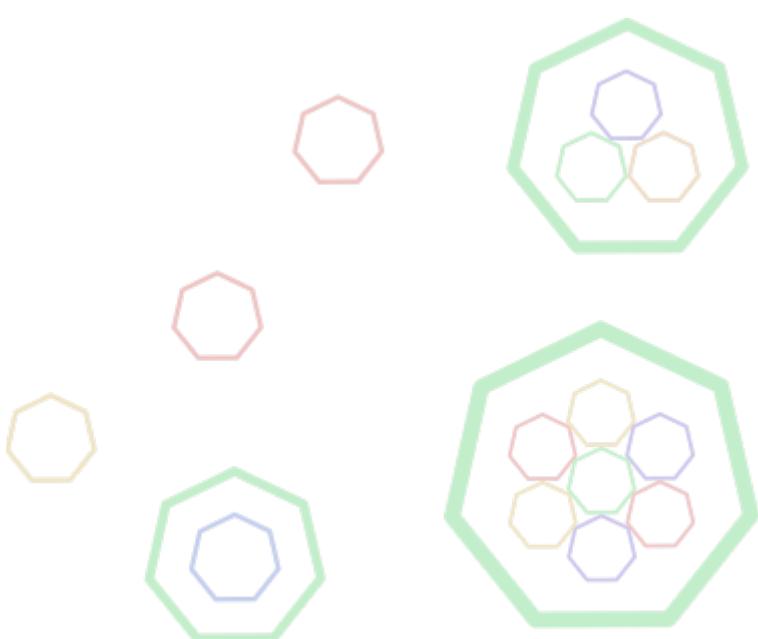
Use the PVC in a Pod:

```
kind: Pod
apiVersion: v1
metadata:
  name: apache
spec:
  containers:
    - name: apache
      image: httpd:2.4
      volumeMounts:
        - name: data
          mountPath: /usr/local/apache2/htdocs/
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: website-data
```



Access Modes

- Based on backend type:
 - Multiple mount point: NFS, CephFS, GlusterFS
 - Single mount point: block storage, regular FS, Cinder, HostPath



Access Modes Types

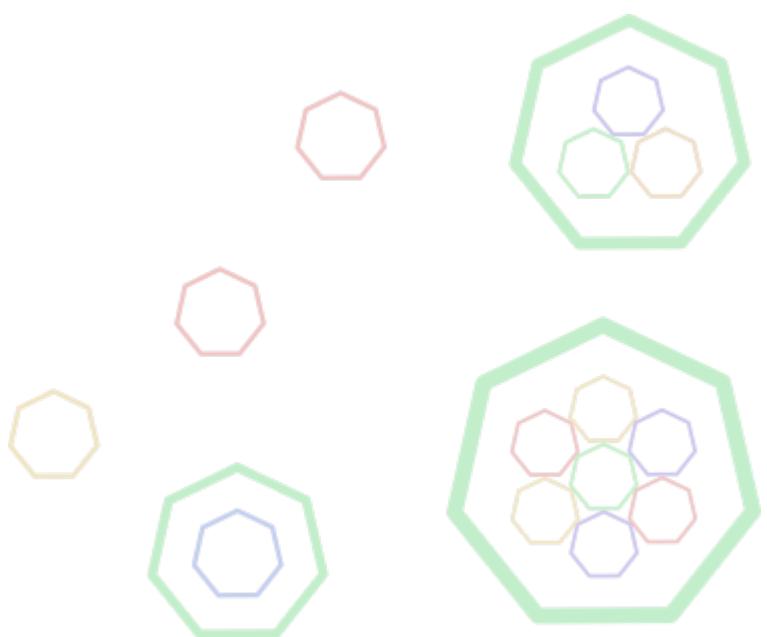
Different types of access modes:

- RWX/ReadWriteMany: Pods on several Nodes may mount this volume in a read/write mode
- ROX/ReadOnlyMany: Pods on several Nodes may mount this volume in a read only mode
- RWO/ReadWriteOnce: Only Pod on the same Node can mount this volume

RWX mode generally allows ROX mode also



All volume plugins allow at least RWO mode

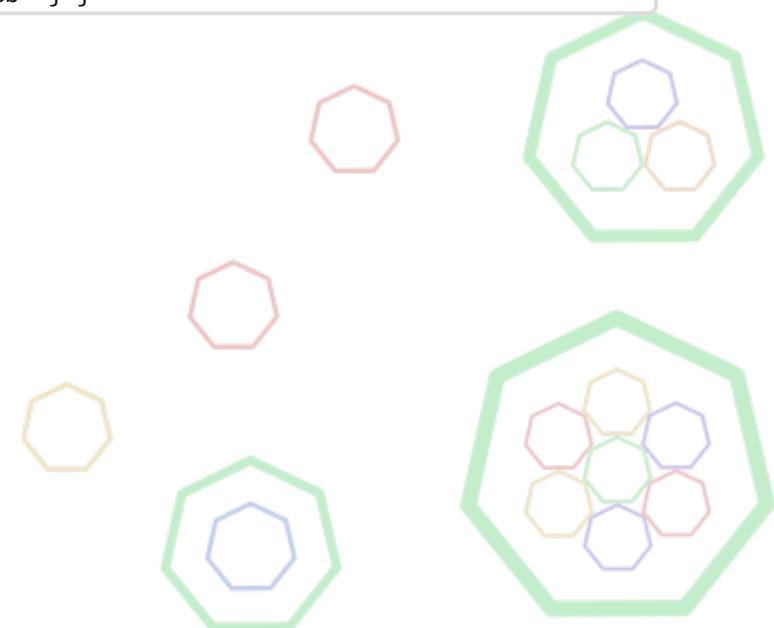


PVC in StatefulSets

In StatefulSets:

- scaling creates numbered Pods (name-0, name-1, ...)
- each Pod receives its own numbered PVC (name-0, name-1, ...)
- volumes are specified as a template:

```
---  
kind: StatefulSet  
apiVersion: apps/v1  
metadata:  
  name: web  
spec:  
  template:  
    metadata: { "labels": { "app": "web" } }  
    spec:  
      containers:  
        - name: nginx  
          image: nginx  
          volumeMounts:  
            - name: data  
              mountPath: /mnt  
      volumeClaimTemplates:  
        - metadata:  
            name: data  
          spec:  
            accessModes: [ "ReadWriteOnce" ]  
            resources:  
              requests:  
                storage: 1Gi  
      replicas: 2  
      selector: { "matchLabels": { "app": "web" } }
```

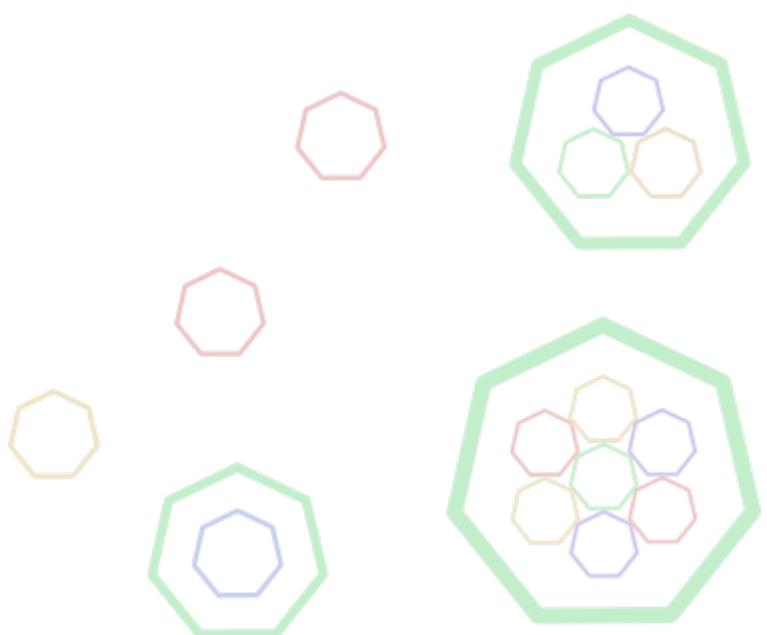


StorageClasses — Principle

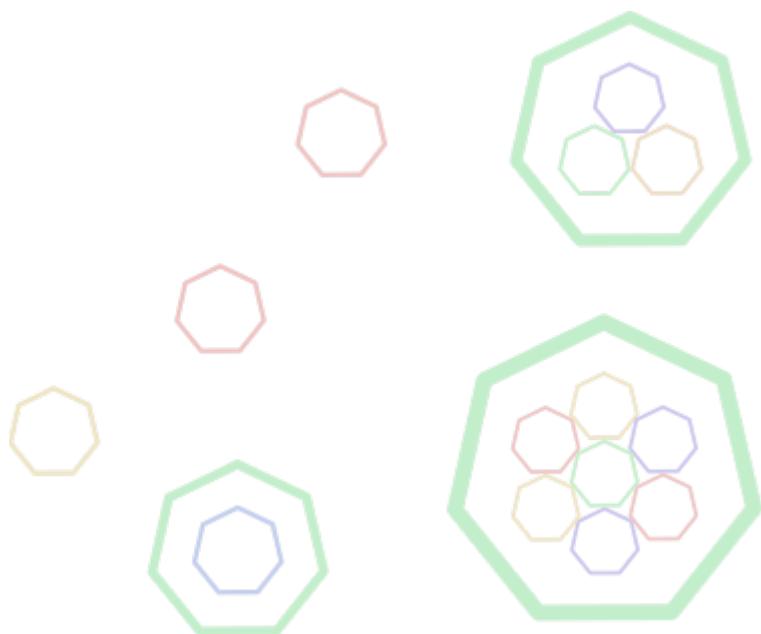
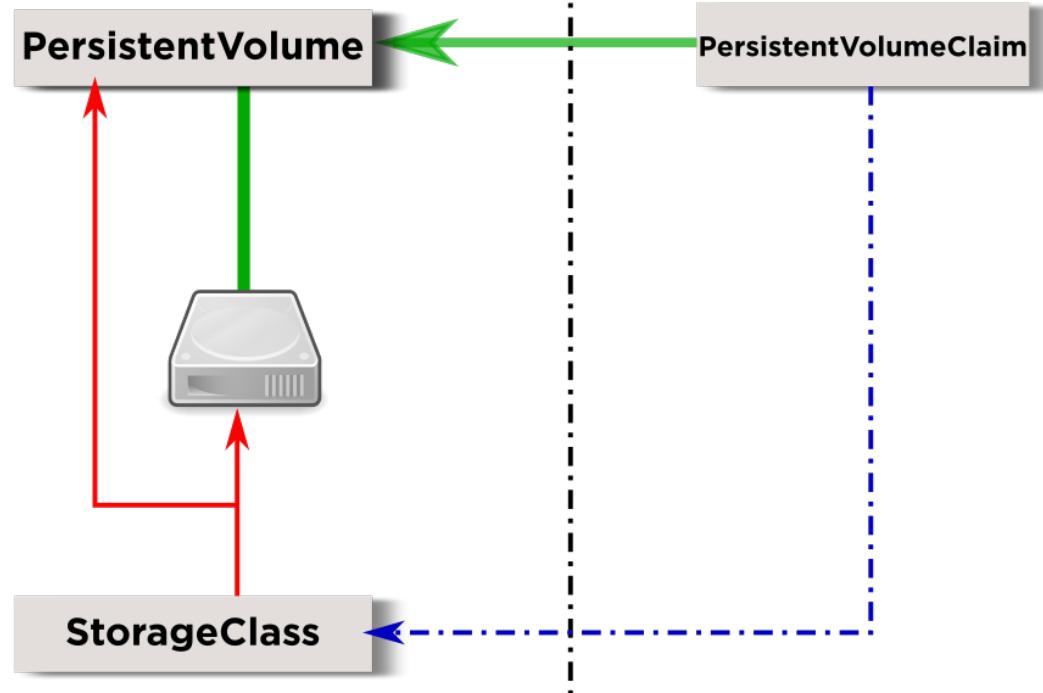


No standard Controller provided

- Defined by cluster administrators
- Convention/interface between devs and ops
- Abstraction of volume features, should describe the class:
 - volume performance: ssd, rotative disk
 - volume backup feature: incremental backup, daily/weekly backed up
 - cost based, throughput/Bandwidth
- Storage classes enable dynamic provisionning of volumes



Infrastructure User

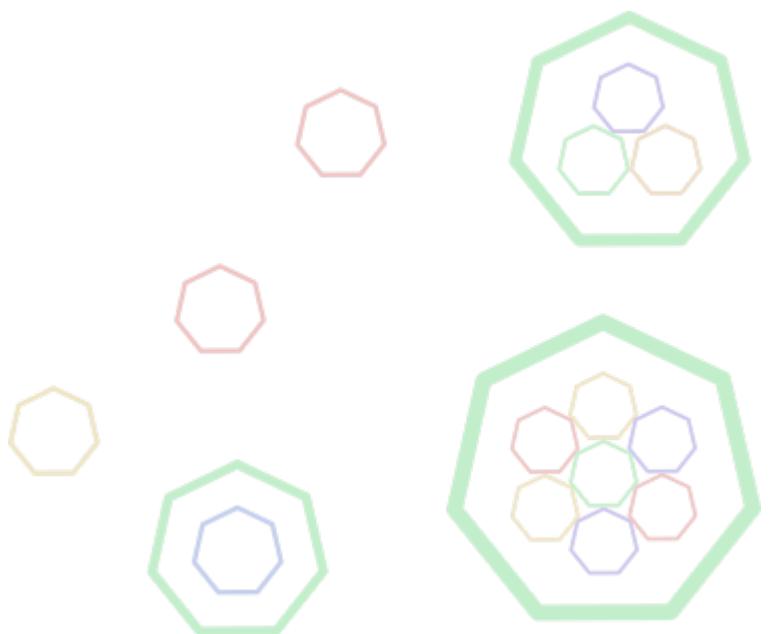


Demo: Deploying a StorageClass Controller



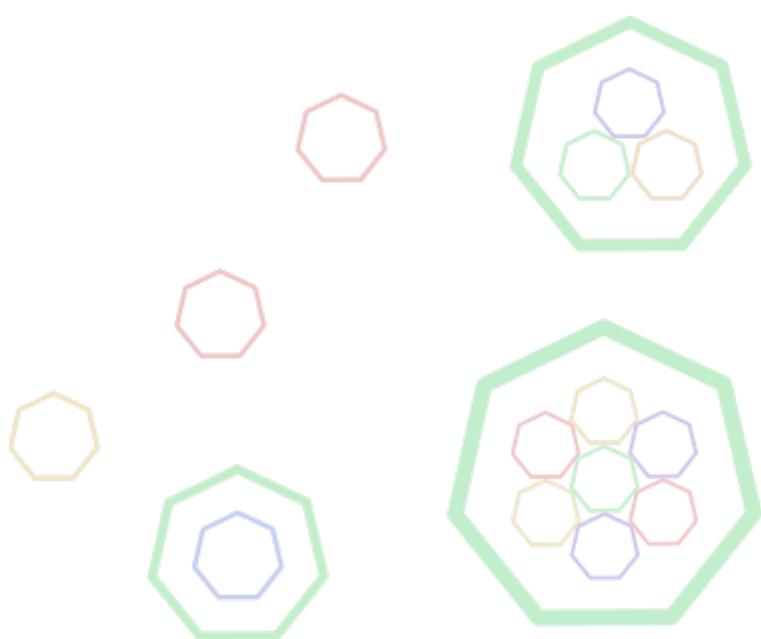
Instructor will:

- Deploy a new StorageClass Controller on the cluster
- List it
- Demonstrate how to use it for a new PVC



Volume Capacity

- Devs can request specific size
- Kubernetes will bind a PV that fulfills capacity and other parameters
- Each PV has a specific size
- Kubernetes will choose a volume that has *at least* the requested size



Auto Allocation/Provisioning

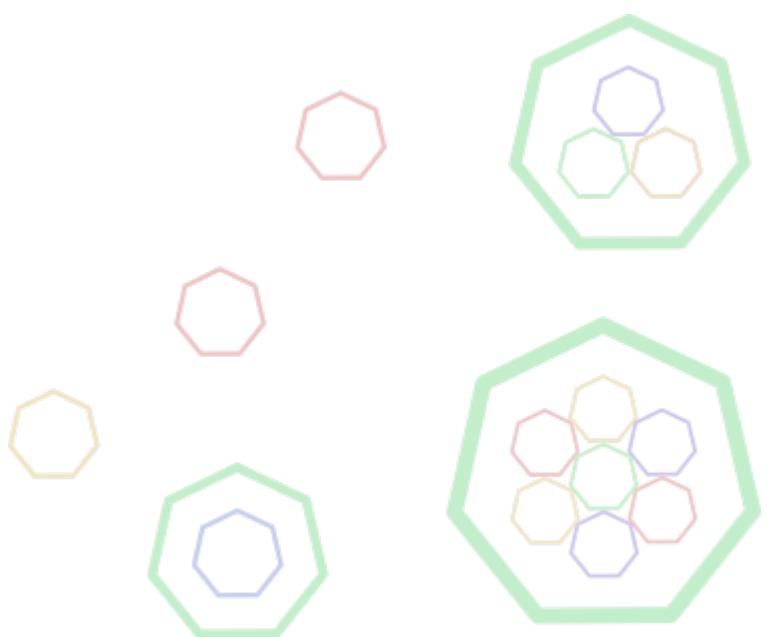
Kubernetes tries to use existing volumes first

Static provisioning:

- Cluster admins need to create volumes first
- Provision a pool of volume to be consumed by devs
- Monitor pool size

Dynamic provisioning:

- Cluster admin configures dynamic provisioning based on underlying implementation (using e.g. StorageClasses + IaaS-specific Controller)
- Monitor resources or billing



Lab 25.1: Use Volumes

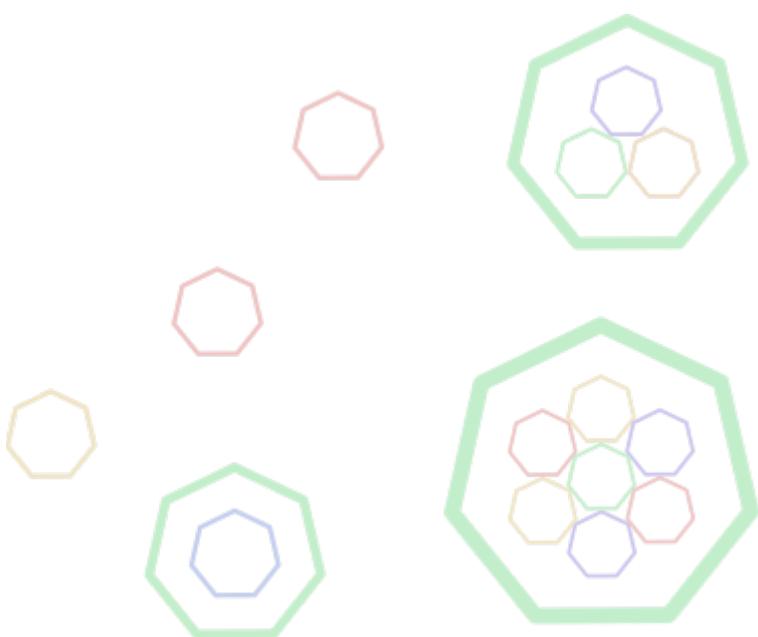


Objective

- Claim a Volume and use it

Steps

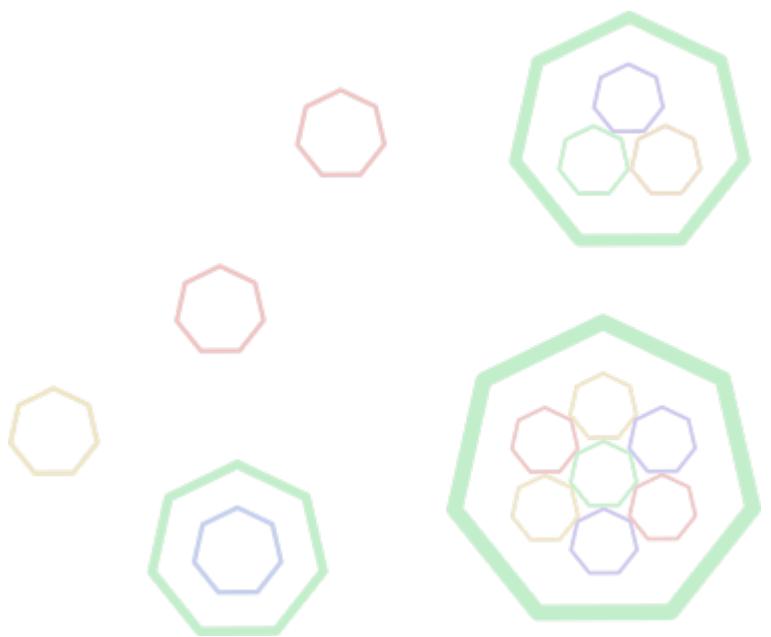
- Create a PersistentVolumeClaim with a RWO mode and 10M Capacity
- Mount this PVC on a Pod running a busybox/debian/ubuntu image
- Create files in the volume
- Delete Pod and recreate it
- Check that files in the Volume still exist



Checkpoint: Data Management

■ Can we attach a RWO Volume to the consumer Deployment?

- yes
- no



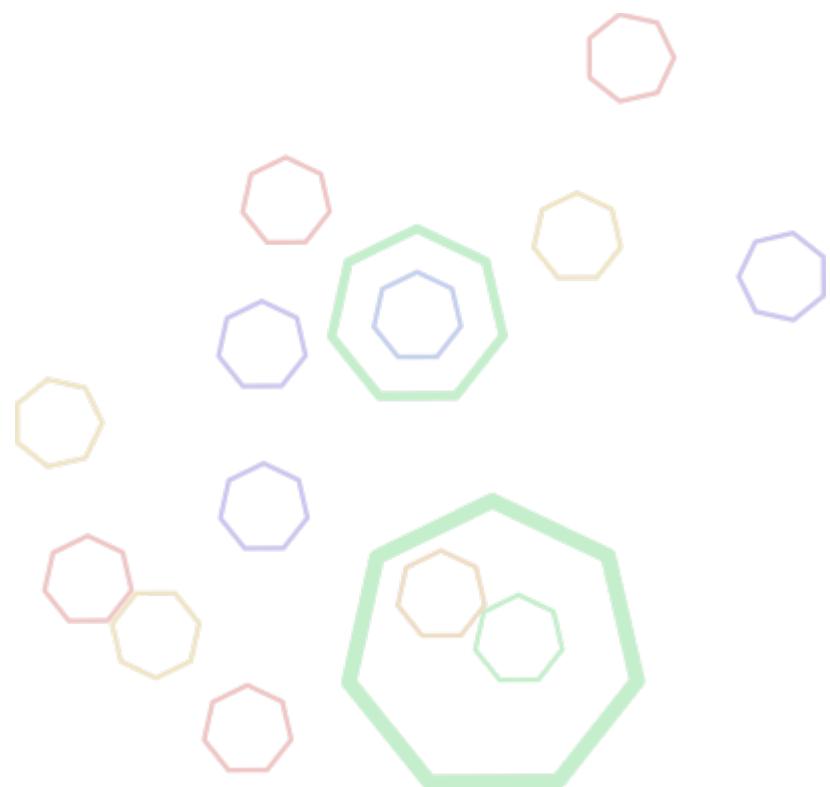
CONFIGMAPS & SECRETS

Lesson 26: ConfigMaps & Secrets

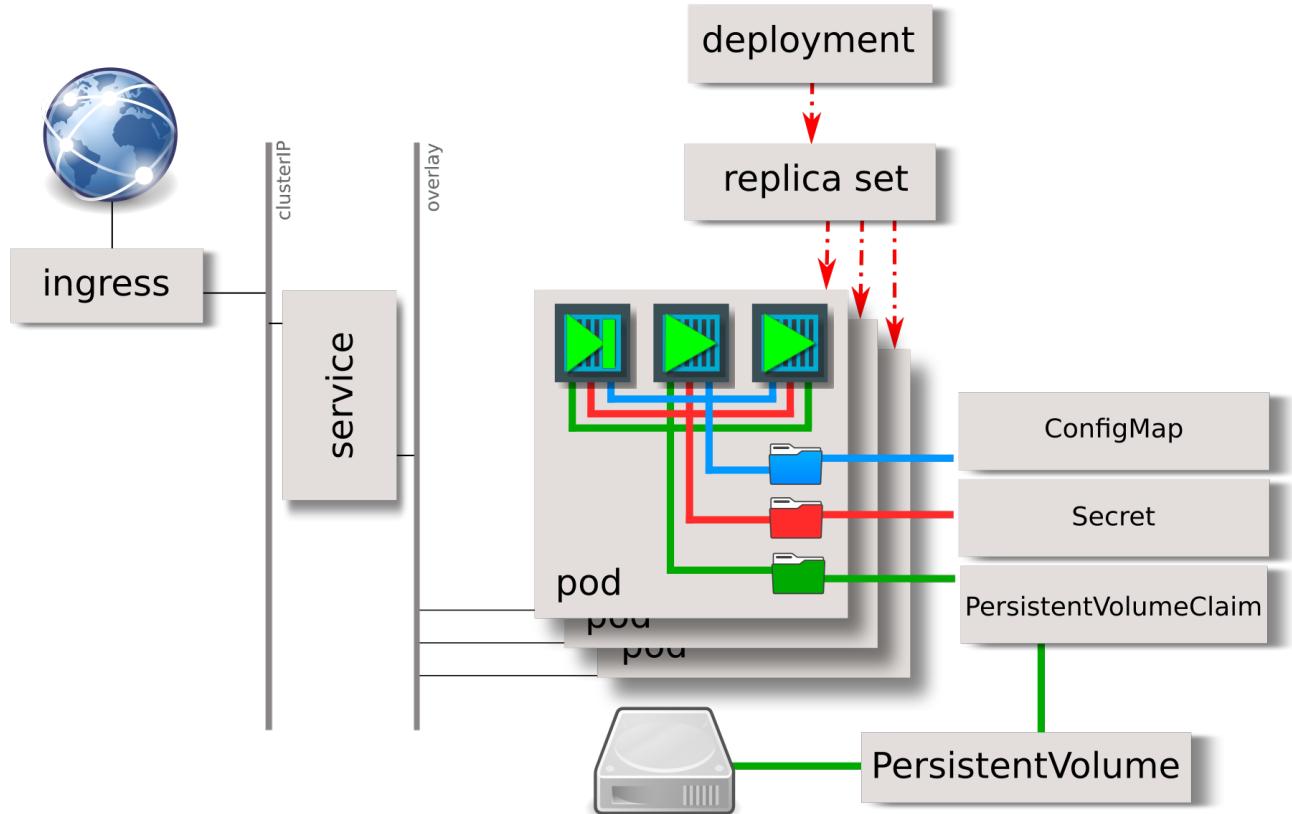
Objectives

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

- understand how to tune an application's configuration
- manage configuration as ConfigMaps
- properly handle secrets in applications



API Objects Overview



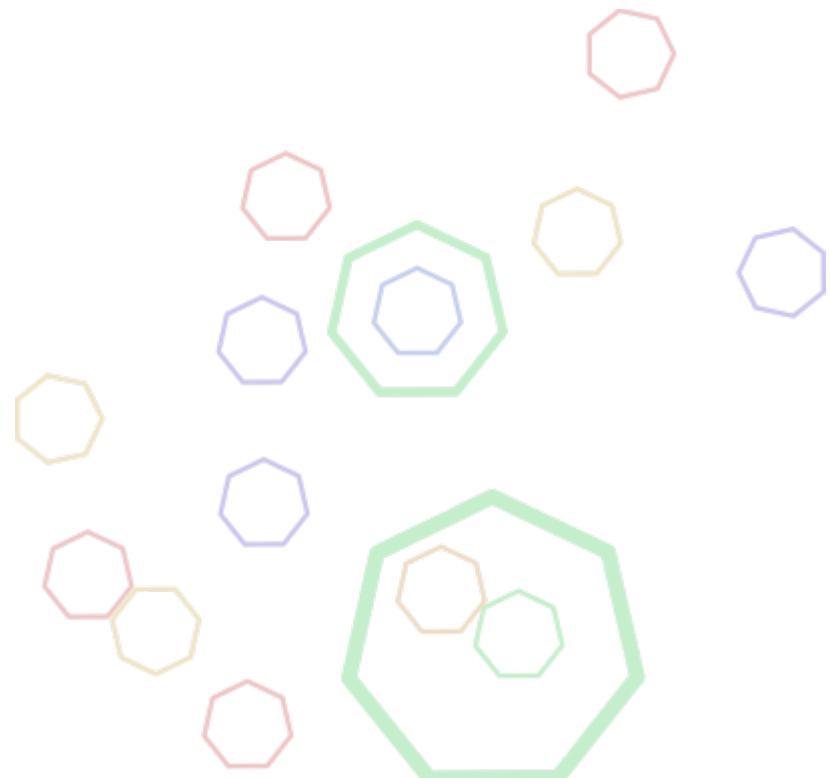
Application Components and Lifecycle

Applications can be divided in three parts:

- **Code**: doesn't change between dev and prod
 - ⇒ should be immutable artifact (container image)
- **Configuration**: changes from dev to prod but not over time
 - ⇒ should be injected at runtime (as arguments)
- **Data**: changes over time
 - ⇒ should be stored aside

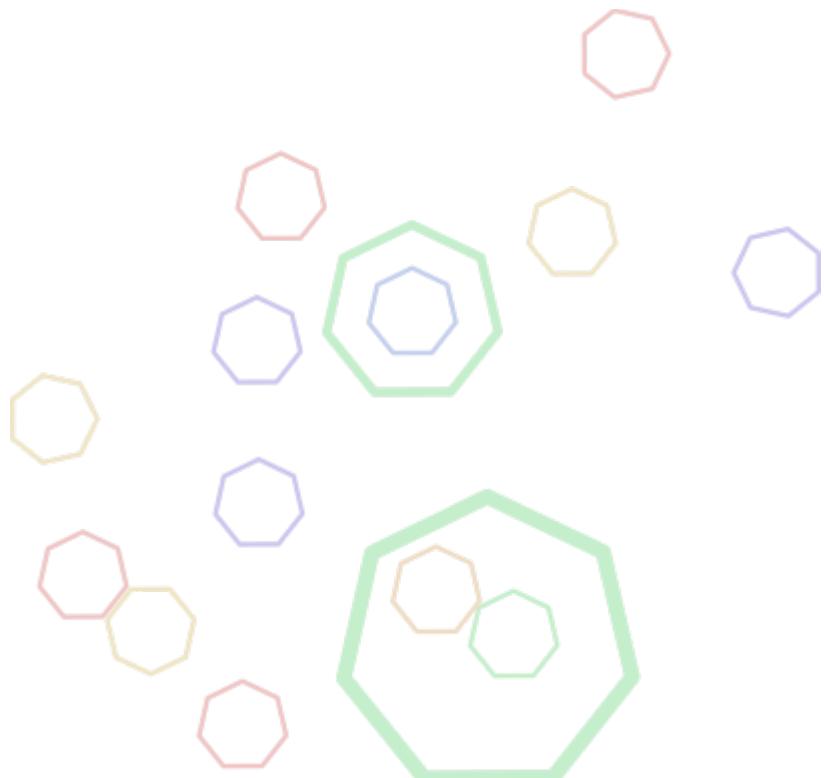


Data and configuration can be stored in Volumes but
Kubernetes offers a better way to handle configuration:
ConfigMaps



ConfigMap — Principle

- Designed to store small amount of data
- Avoids using configuration image and helps usage of upstream images
- Single value, file or multiple files
- Collection of key/value pairs: no hierarchy
 - key: filename
 - value: file content or literal value



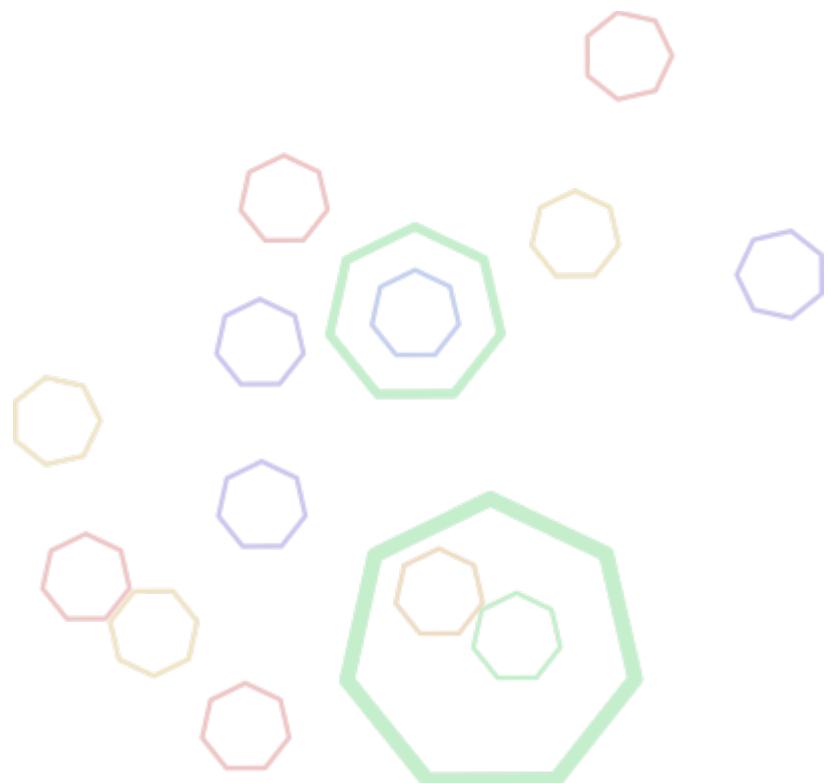
Creating a ConfigMaps from literal

Specify key and value on command line:

```
$ kubectl create configmap app1-config --from-literal=title=demo
```



Multiple key/value pairs can be specified



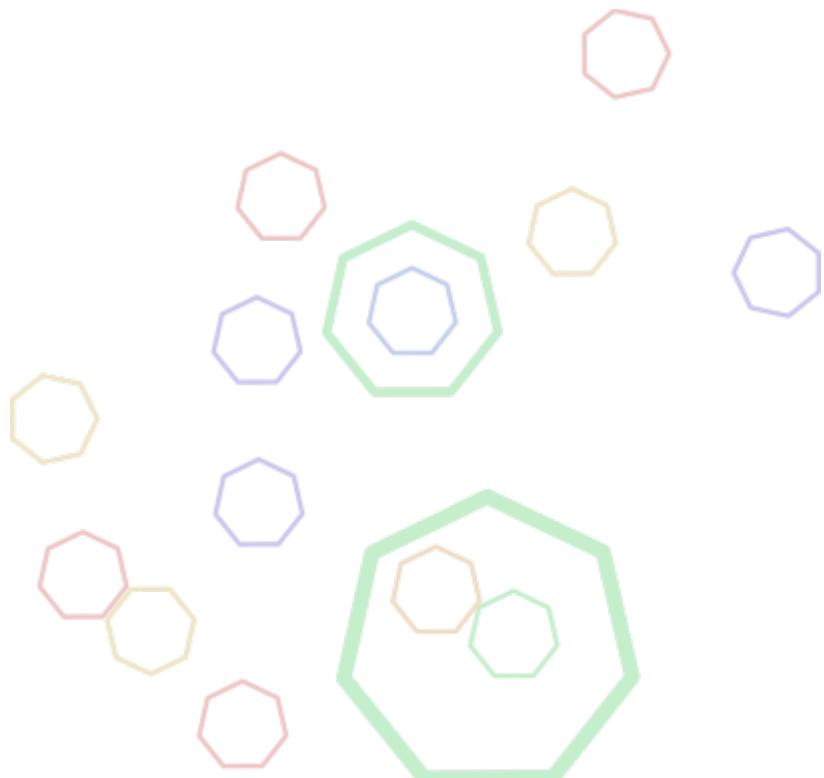
Creating ConfigMaps from config files

Use the `--from-file` option:

```
$ kubectl create configmap app1 --from-file=app1.properties  
$ kubectl create configmap app2 --from-file=config/app2/
```

- Sub directories are not used
- Use filename as key
- Override key with:

```
$ kubectl create configmap <configmap name> --from-file=<new key>=<path to file>
```

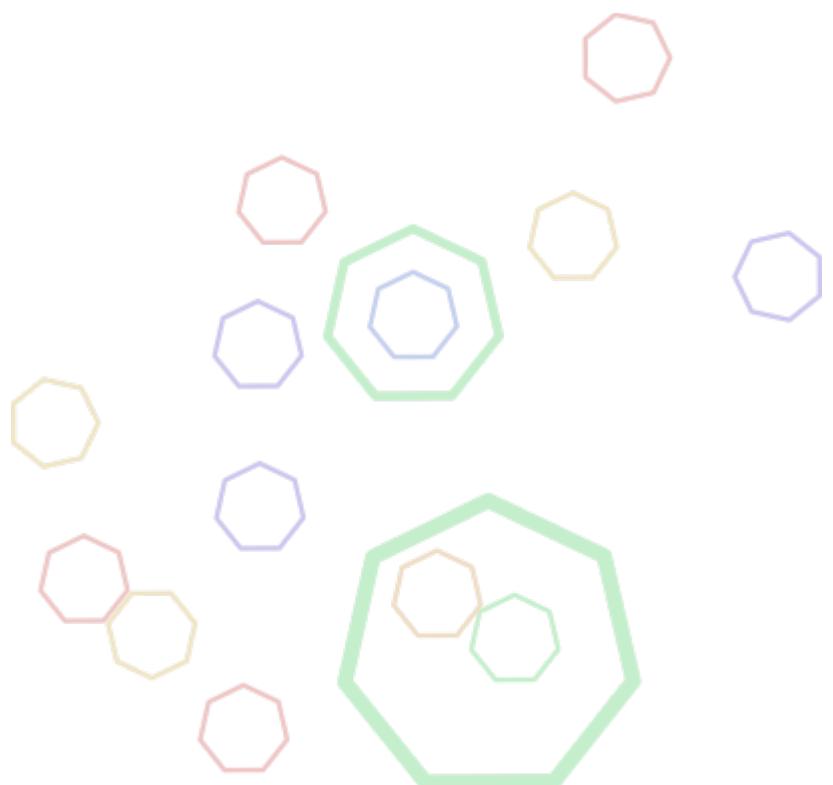


Creating ConfigMaps from YAML manifest

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app1
data:
  app1.properties: |
    title=demo
    width=50
    height=30
```

Then create the ConfigMap with `kubectl`:

```
$kubectl apply -f <manifest>.yaml
```

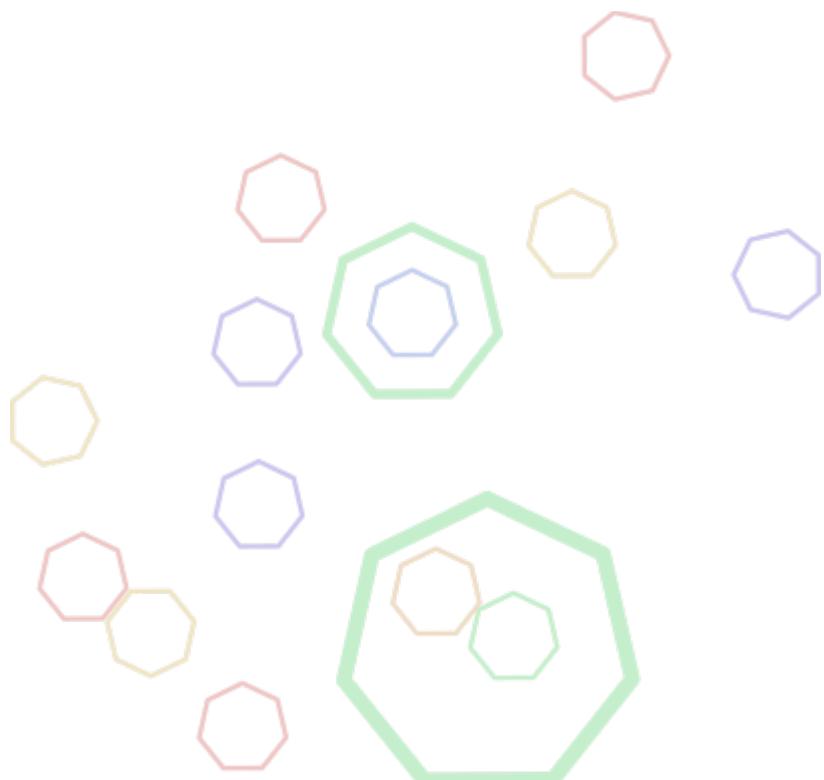


Using ConfigMaps

Consumed by containers:

- mounted as files (l20)
- or injected as environment variables (l12)

```
kind: Pod
apiVersion: v1
metadata:
  name: app1
spec:
  containers:
    - name: app
      image: app
      env:
        - name: LOG_LEVEL
          valueFrom:
            configMapKeyRef:
              name: app1-log-level
              key: log_level
      volumeMounts:
        - name: app-configs
          mountPath: /etc/app
  volumes:
    - name: app-configs
      configMap:
        name: app1-configs
```



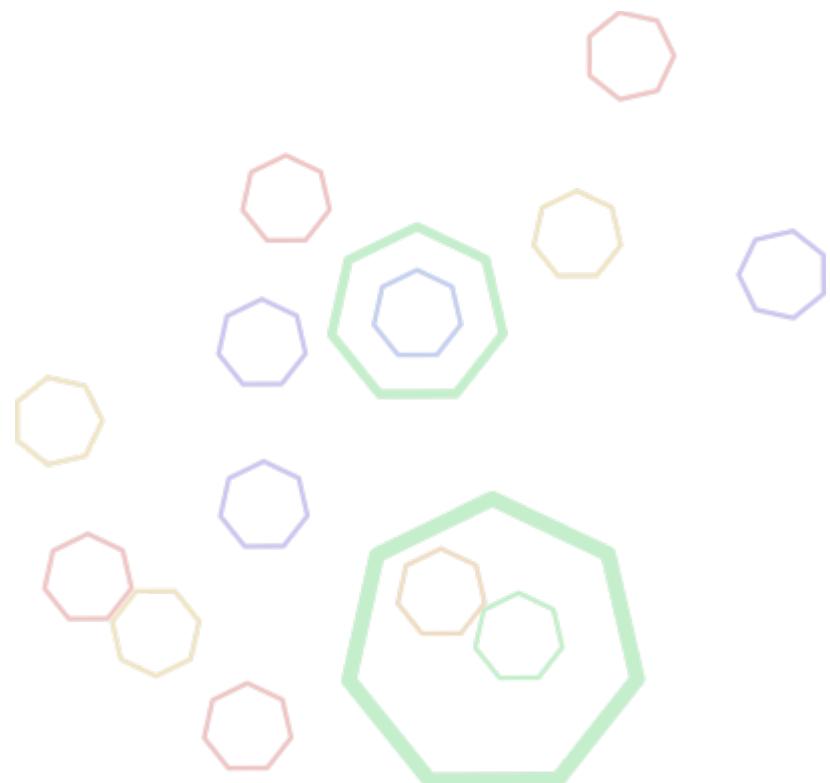
Import multiple keys from config file

- `--from-env-file` loads multiple key/values from one files
- useful for multiple environment variables:

```
$ cat env.conf
LOG_LEVEL=debug
HEIGHT=30
WIDTH=50
$ kubectl create configmap app1-env --from-env-file env.conf
```

is equivalent to creating a ConfigMap with:

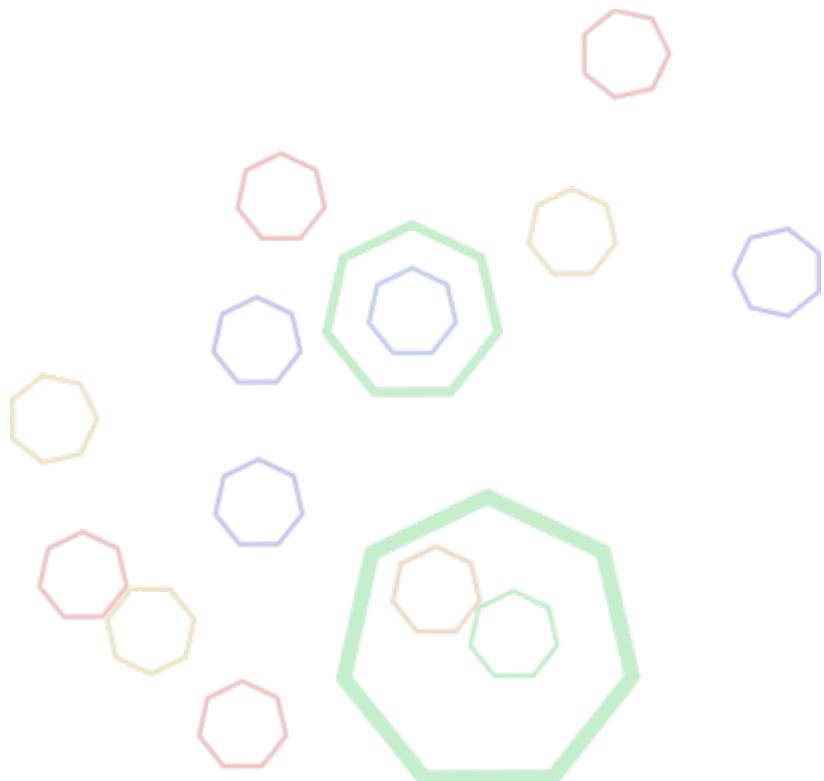
```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app1-env
data:
  LOG_LEVEL: debug
  HEIGHT: 30
  WIDTH: 50
```



Loading multiple environment variables from ConfigMap

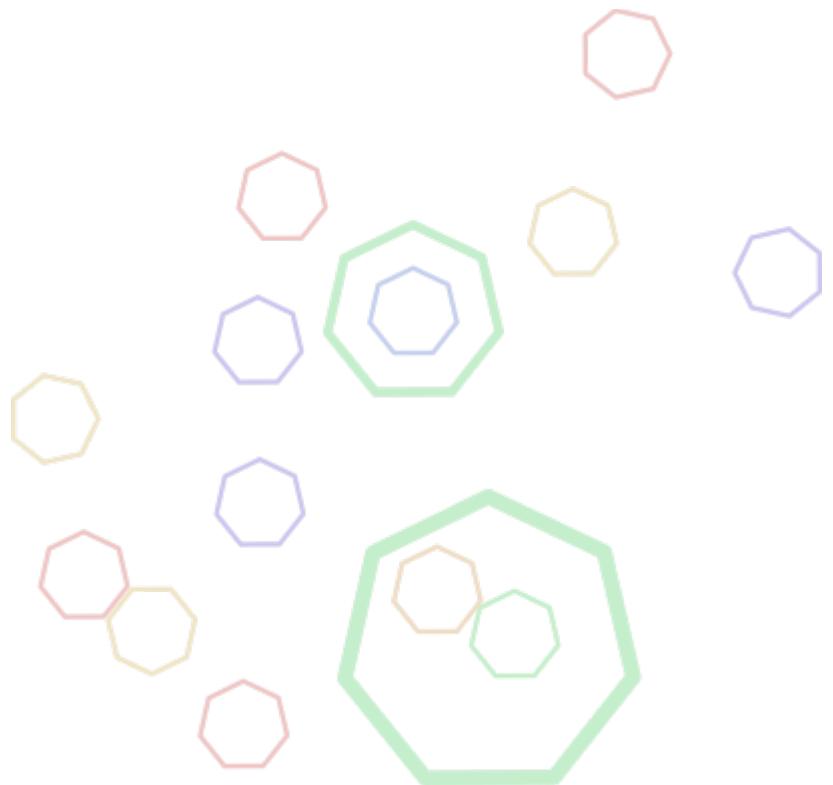
The `envFrom` YAML key injects one environment variable per hash key in the ConfigMap:

```
kind: Pod
apiVersion: v1
metadata:
  name: app1
spec:
  containers:
    - name: app
      image: app
      envFrom:
        - configMapRef:
            name: app1-env
```



Secrets

- Sensitive data (tokens, SSH keys, passwords, etc.) must be handled with care
- Never stored to a non-volatile storage
- Consume by Pod to pull image or at runtime
- Used in Ingress for TLS



Creating Secrets from literal or app

- Similar to ConfigMaps: collection of key/value pairs
- You need to specify a type:
 - generic
 - docker-registry
 - basic-auth
 - ...

From literal

```
$ kubectl create secret generic app-pass --from-literal=password=<app_password>
```

From file

```
$ kubectl create secret generic app-password --from-file=app-password.txt
```

Creating Secrets from YAML manifest

YAML manifest uses a base64 encoding to hide secret content:

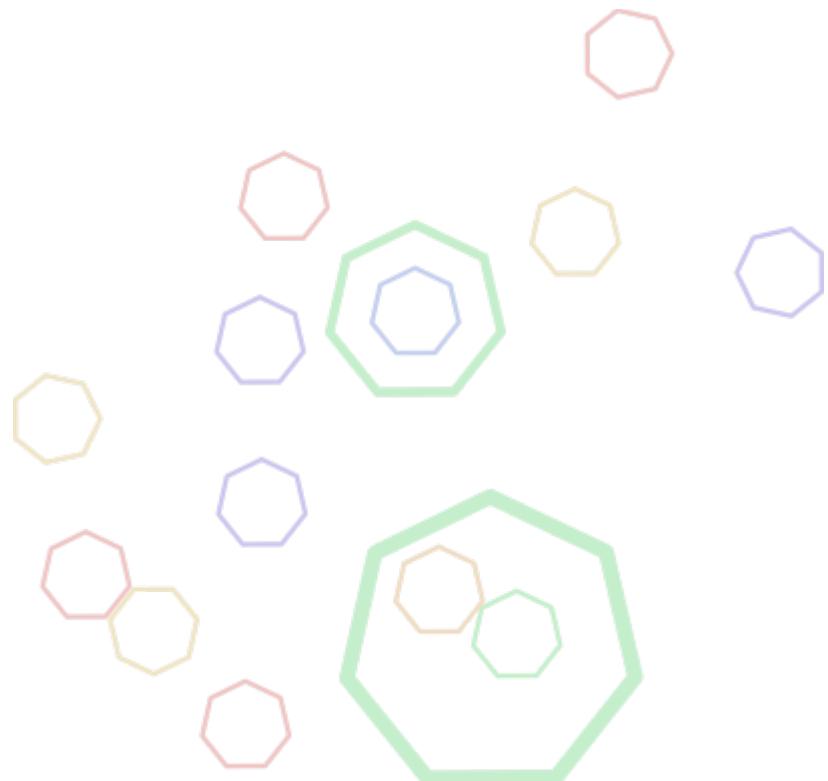
```
kind: Secret
apiVersion: v1
metadata:
  name: my-password
type: Opaque
data:
  password: cmVhbFBheHh3b3JkCg==
```

You can also use `stringData` with no encoding:

```
kind: Secret
apiVersion: v1
metadata:
  name: my-password
type: Opaque
stringData:
  password: realPaxxword
```



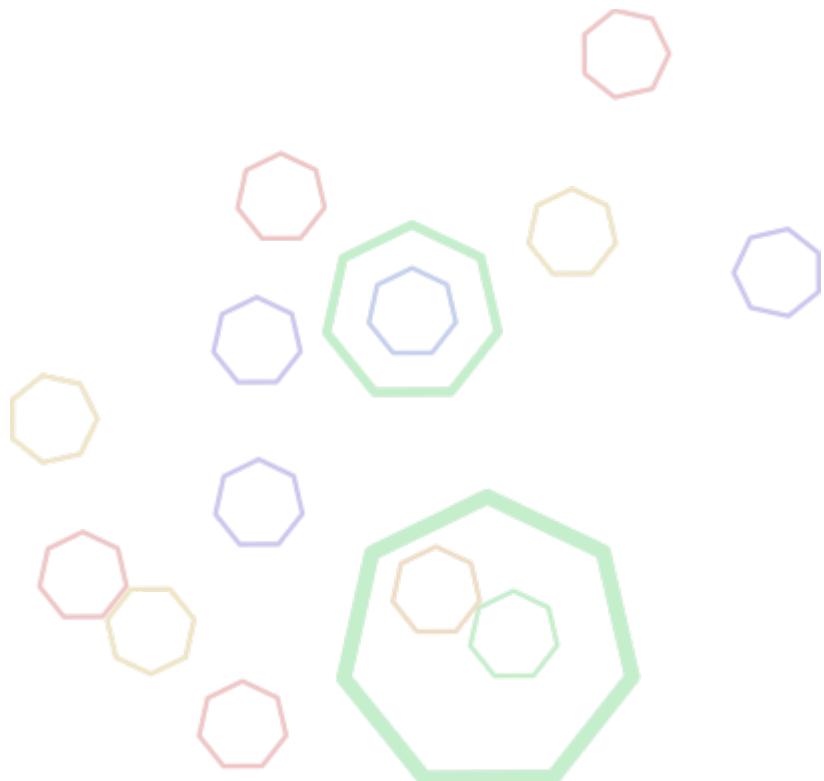
To retrieve Secret value, use `kubectl get -o yaml secret <secret_name>` (or `x` in K9s)



Using Secrets

- Mounted as files or used as environment variables
- Secret is a Volume type
 - `mountPath` is a non-existing directory where filenames will correspond to Secret keys
- You can use symlinks to override only one file in an existing directory

```
kind: Pod
apiVersion: v1
metadata:
  name: apache
spec:
  containers:
    - name: apache
      image: httpd:2.4
      volumeMounts:
        - name: password
          mountPath: "/usr/local/apache2/password"
          readOnly: true
  volumes:
    - name: password
      secret:
        secretName: my-password
```



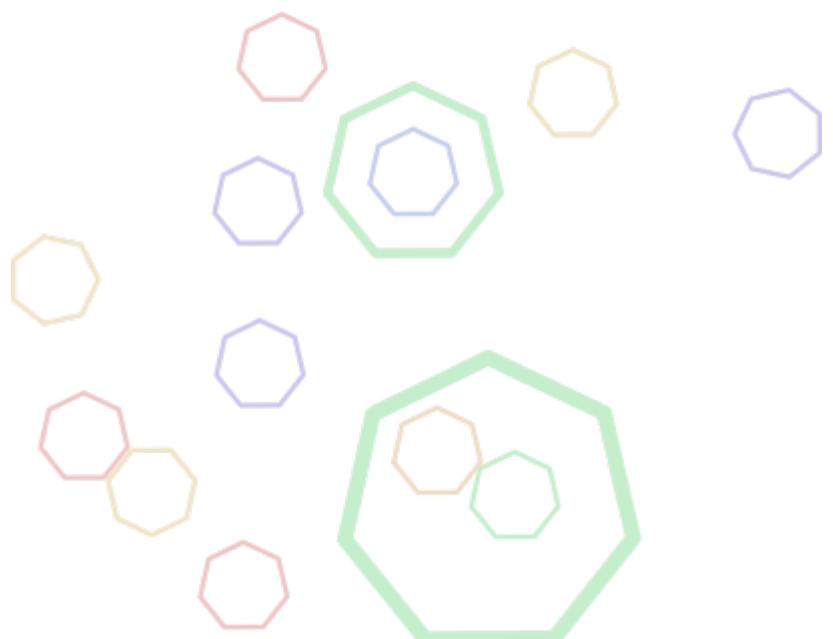
Secrets – Changing File Layout

- You can reorganize volume layout to:
 - Change filenames
 - Create directory hierarchy

```
volumes:  
- name: password  
  secret:  
    secretName: my-password  
    items:  
      - key: password  
        path: htpasswd  
      - key: username  
        path: usernames/app/username
```

results in:

```
$ tree /usr/local/apache2/password  
/usr/local/apache2/password  
├── htpasswd  
└── usernames  
    └── app  
        └── username  
  
2 directories, 2 files
```



Secret – Filesystem Rights

- Specify mode for all files:

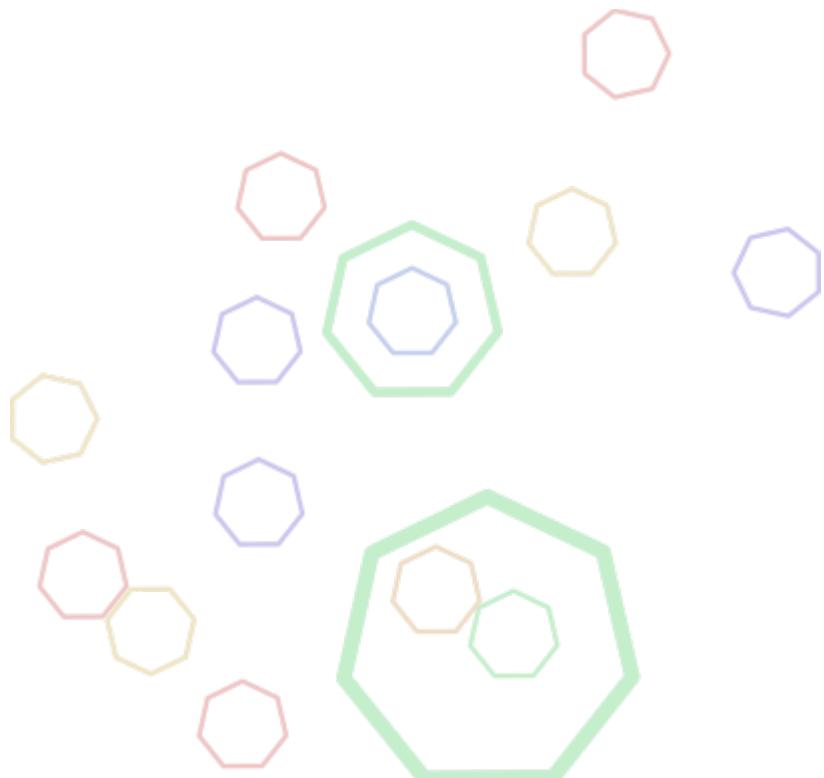
```
volumes:  
- name: password  
  secret:  
    secretName: my-password  
    defaultMode: 256 # 0400
```

- Specify mode for specific file:

```
volumes:  
- name: password  
  secret:  
    secretName: my-password  
    items:  
    - key: password  
      path: htpasswd  
      mode: 511 # 0777
```



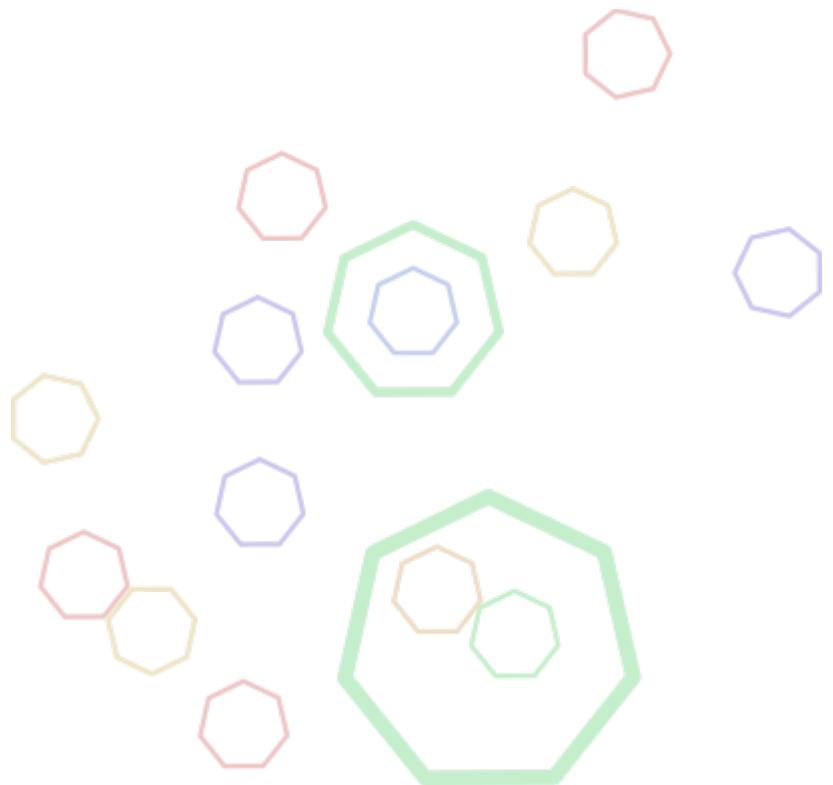
Rights are specified in decimal notation



Using Secrets as Environment Variables

Works exactly like ConfigMaps:

```
kind: Pod
apiVersion: v1
metadata:
  name: apache
spec:
  containers:
  - name: apache
    image: httpd:2.4
  env:
    - name: SECRET_PASSWORD
      valueFrom:
        secretKeyRef:
          name: my-password
          key: password
```



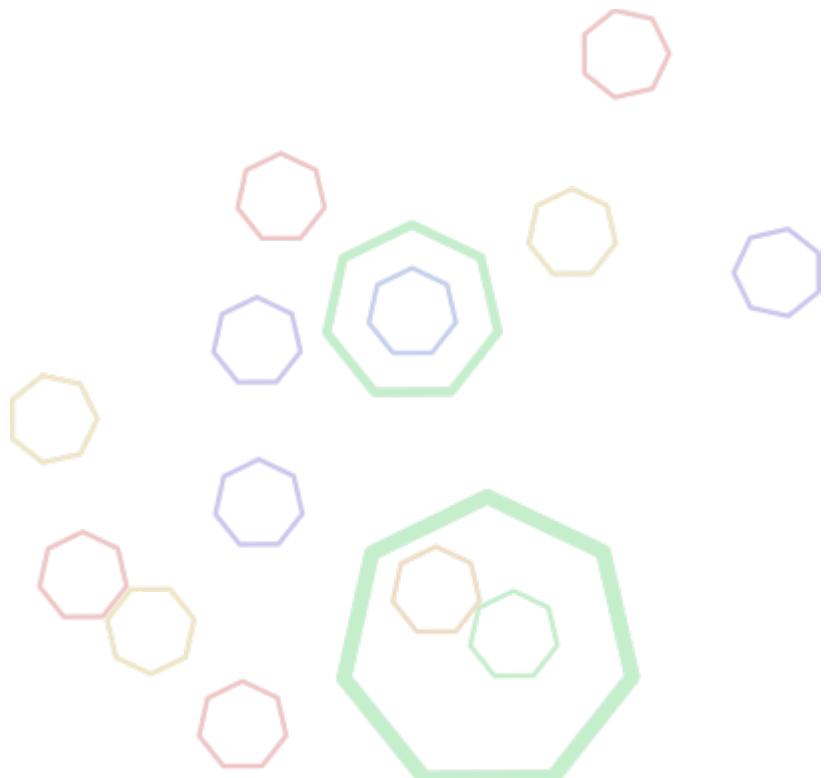
Using Secrets to pull Images

- Create a Docker Registry Secret:

```
$ kubectl create secret docker-registry myregistrykey --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
```

- Reference secret in Pod definition

```
kind: Pod
apiVersion: v1
metadata:
  name: apache
spec:
  containers:
    - name: apache
      image: camptocamp/httpd_optimized:1.3.45
  imagePullSecrets:
    - name: myregistrykey
```



Lab 26.1: Change producer data

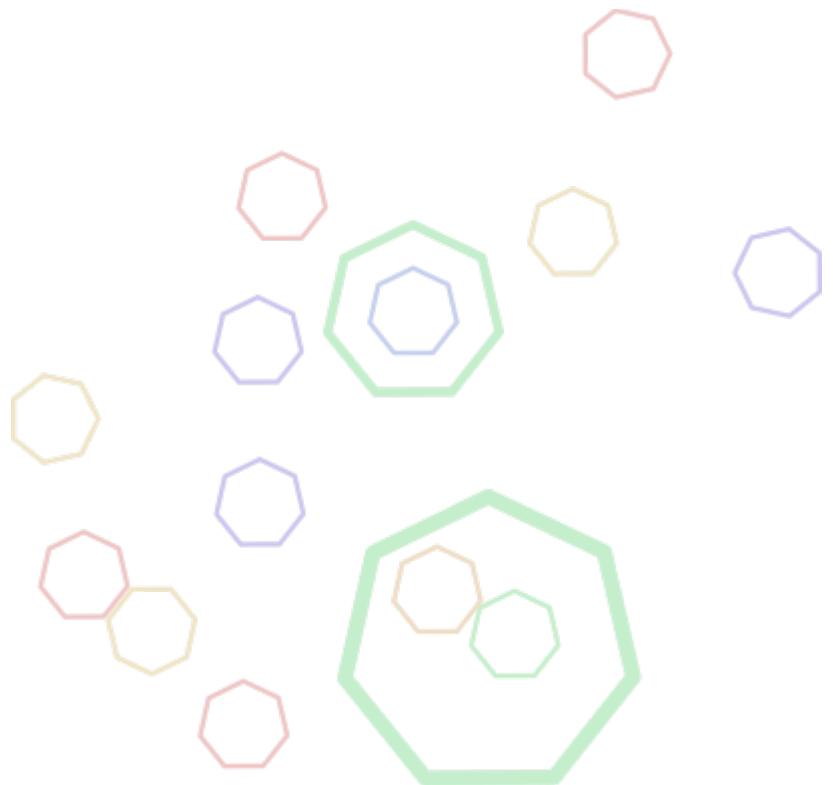


Objective

- Modify data used by backend

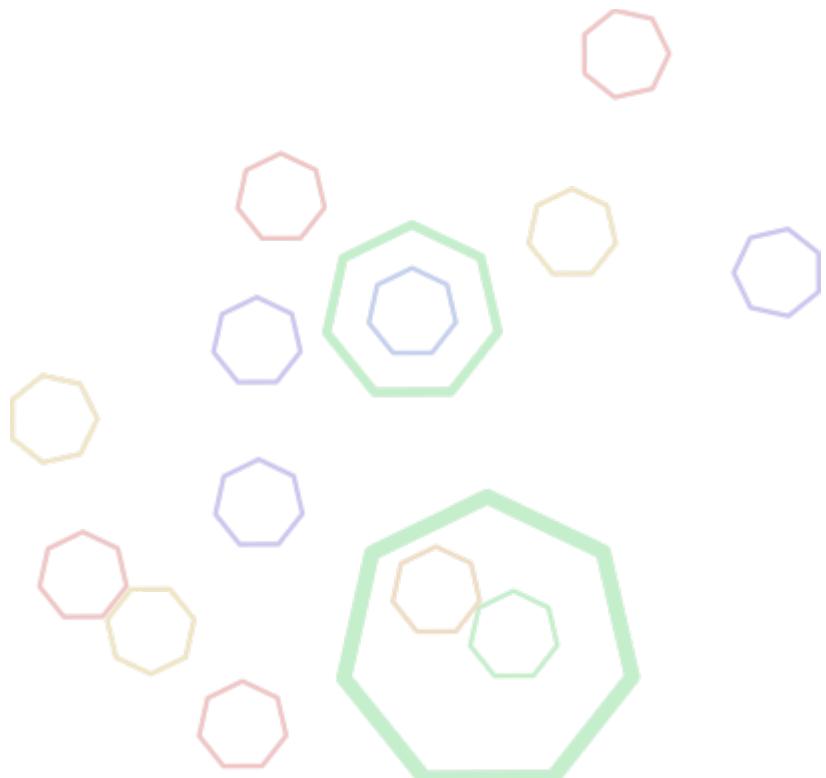
Steps

- Create a *ConfigMap* with the new JSON data
- Update *Deployment* to use the *ConfigMap* (backend loads data from `/etc/backend/data.json`)



Checkpoint: ConfigMaps & Secrets

- Can we use the same ConfigMap for multiple Pods?
 - yes
 - no
- Can we use multiple ConfigMaps in the same Pod?
 - yes
 - no
- ConfigMap can be created from:
 - file
 - environment variable
 - standard input



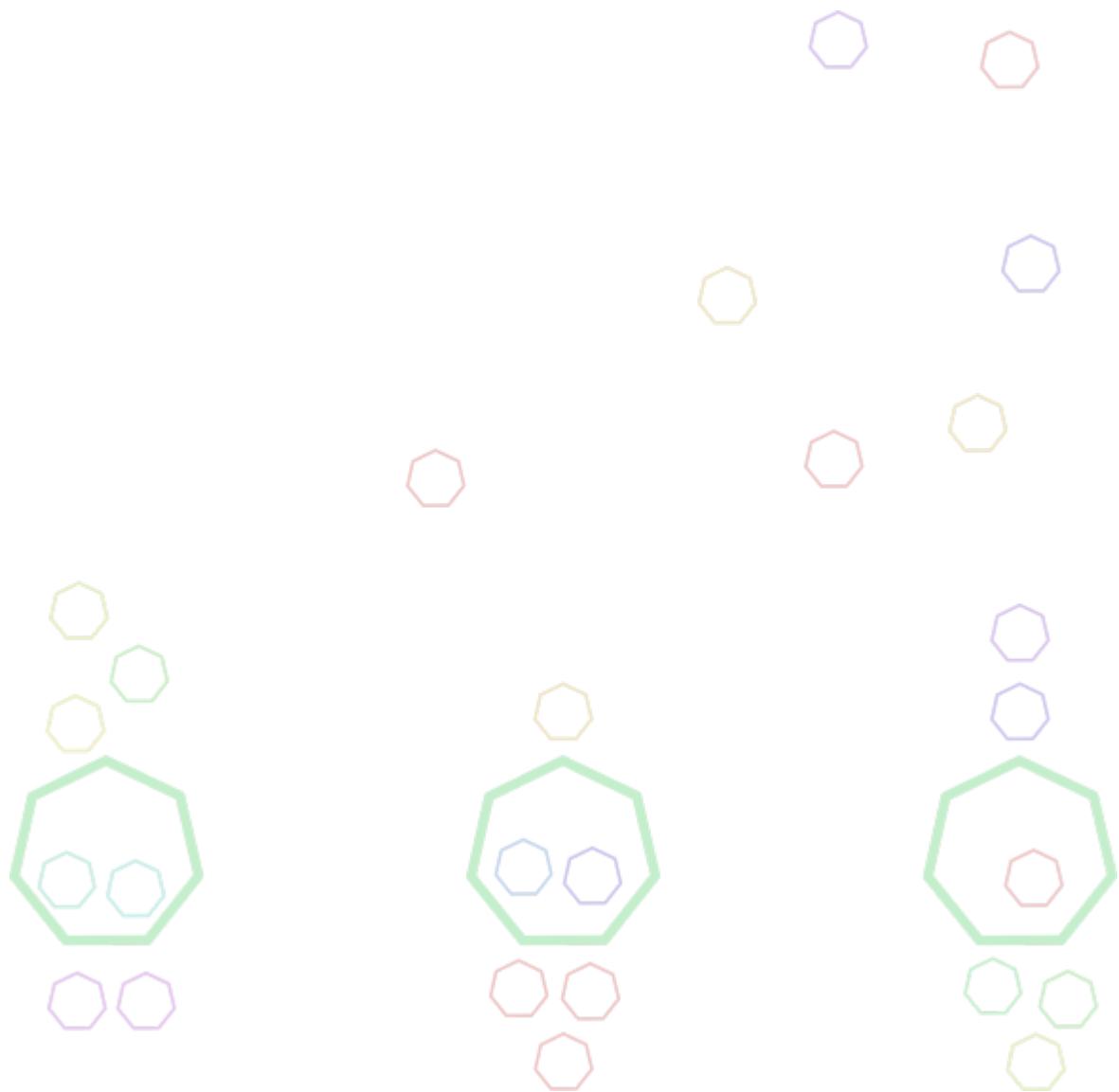
OPERATORS

Lesson 27: Operators

Objectives

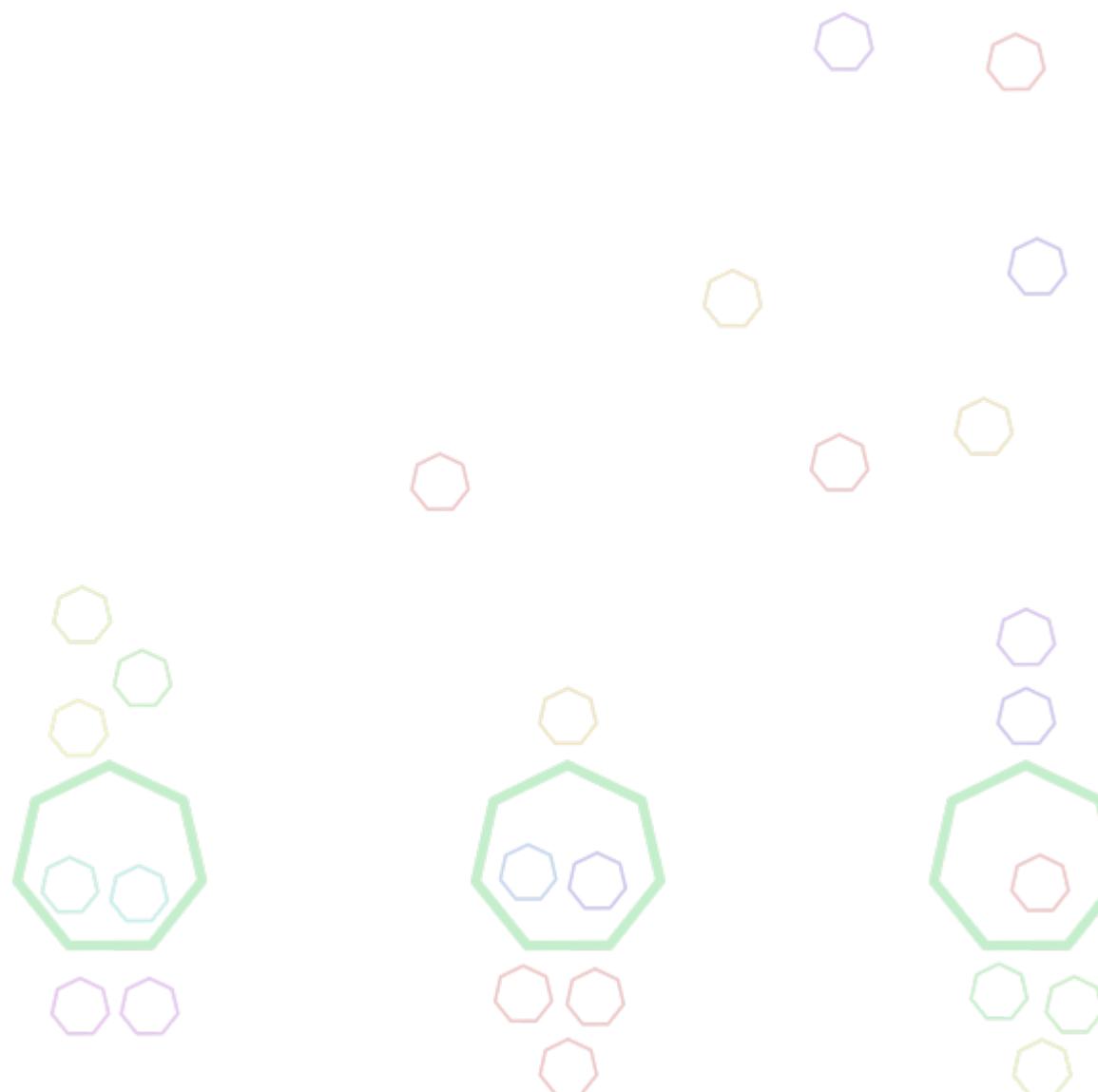
At the end of this lesson, you will be able to:

- Understand what is an operator
- How to use some useful operators
- Deploy CRDs



Operators

- Software Extensions
- Use Custom Resource Definitions (CRDs)
- Use custom controllers



CNCF Projects



Operator: Cert-manager



Manage dynamic certificates in Kubernetes

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: acme-crt
spec:
  secretName: acme-crt-secret
  dnsNames:
  - foo.example.com
  - bar.example.com
  issuerRef:
    name: letsencrypt-prod
    kind: Issuer
    group: cert-manager.io
```



Operator: RabbitMQ



Automate provisioning, management and operations of RabbitMQ clusters

```
apiVersion: rabbitmq.com/v1beta1
kind: RabbitmqCluster
metadata:
  name: rabbitmqcluster-sample
spec:
  rabbitmq:
    advancedConfig: [
      [
        {ra, [
          {wal_data_dir, '/var/lib/rabbitmq/quorum-wal'}
        ]}
      ]
    ].
```

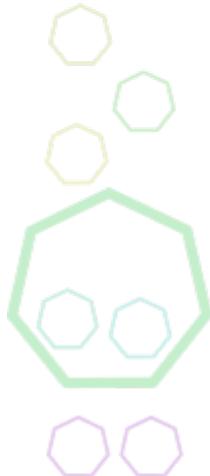


Operator: Redis



Create/configure/manage Redis-failovers

```
apiVersion: databases.spotahome.com/v1
kind: RedisFailover
metadata:
  name: redisfailover
spec:
  sentinel:
    replicas: 3
  redis:
    replicas: 1
  auth:
    secretPath: redis-auth
```

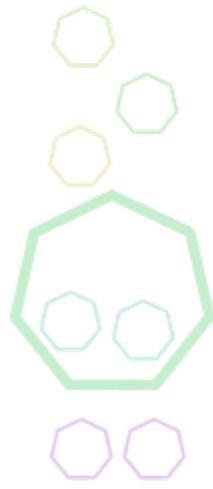


Operator: Vitess



Deploy, scale and manage large clusters of open-source MySQL database instances

```
apiVersion: vitess.io/v1alpha2
kind: VitessCluster
metadata:
  name: aio
  labels:
    app: vitess
spec:
  lockserver:
    metadata:
      name: global
    spec:
      type: etcd2
      etcd2:
        address: etcd-global-client:2379
        pathPrefix: /vitess/global
  cells:
  - metadata:
      name: zone1
    spec:
      lockserver:
        metadata:
          name: zone1
...
...
```

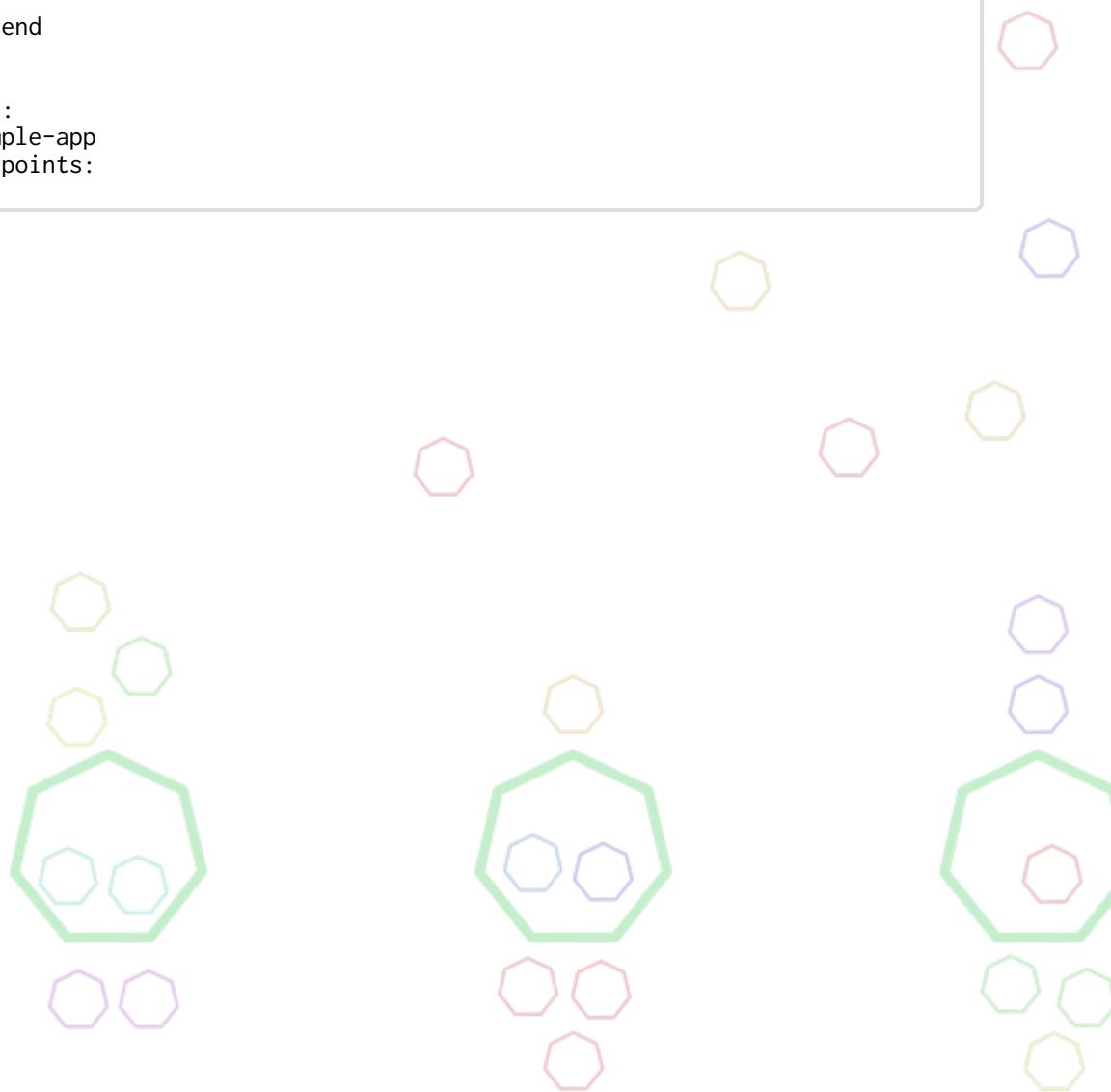


Operator: Prometheus



Deploy and manage Prometheus and related monitoring components

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: example-app
  labels:
    team: frontend
spec:
  selector:
    matchLabels:
      app: example-app
  podMetricsEndpoints:
  - port: web
```

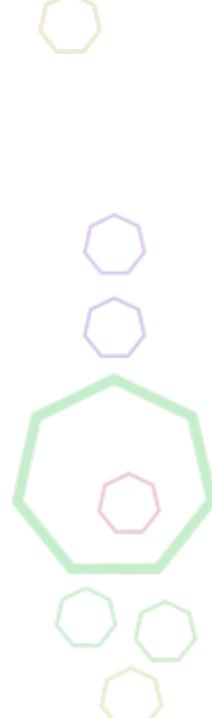
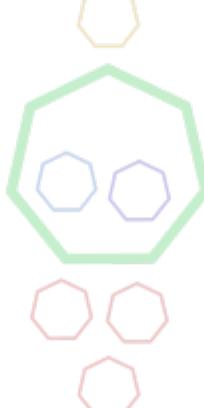
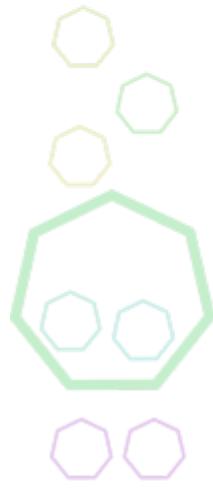


Operator: ArgoCD



Manage ArgoCD clusters

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: initial-repositories
spec:
  initialRepositories: |
    - type: helm
      url: https://storage.googleapis.com/istio-prerelease/daily-build/master-latest-d
      name: istio.io
    - type: git
      url: https://github.com/argoproj/argocd-example-apps.git
```



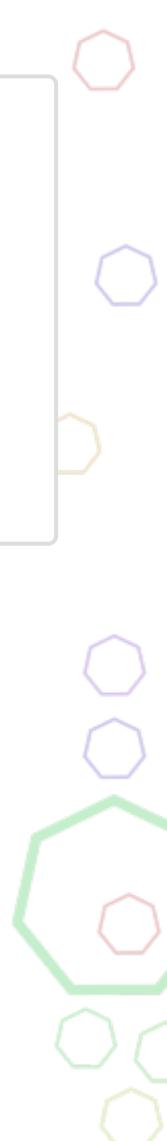
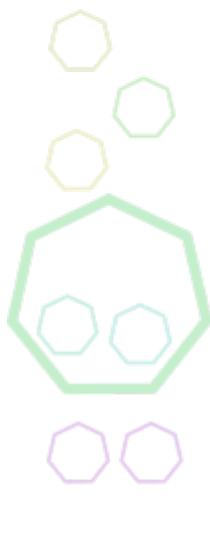
Crossplane



Crossplane

- Extends the principle of Storage Classes to many other resource types (databases, buckets, etc.)
- Cluster operators can create compositions (xRD)
- Developers can create claims (xRC)
- Plugs into Kubernetes RBAC system

```
apiVersion: example.org/v1alpha1
kind: MySQLInstance
metadata:
  namespace: default
  name: example
spec:
  parameters:
    location: au-east
    storageGB: 20
    version: "5.7"
  compositionSelector:
    matchLabels:
      purpose: example
      provider: azure
  writeConnectionSecretToRef:
    name: example-mysqlinstance
```

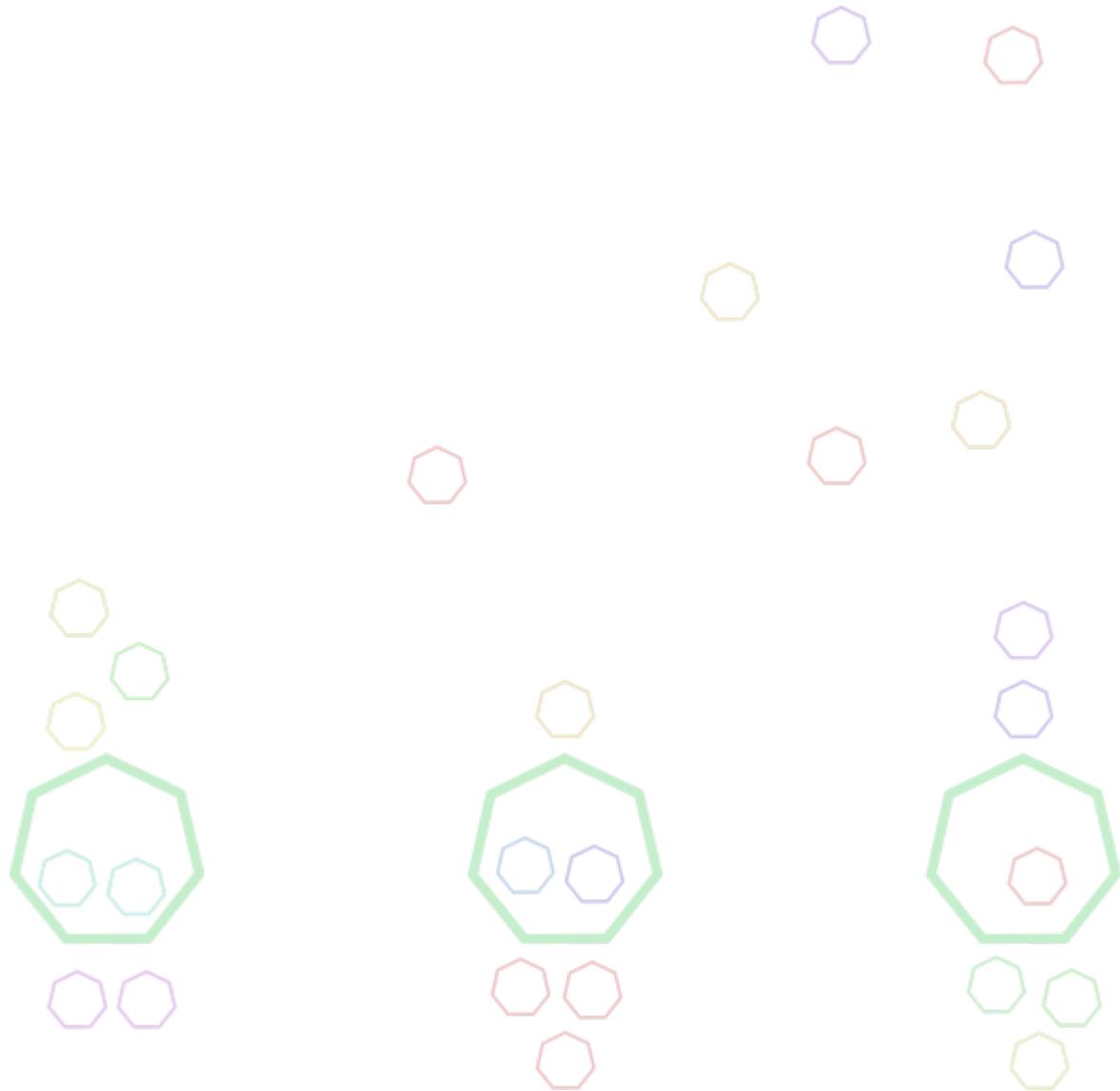


Checkpoint: Operators

- What can operators be used for?
 - Create Deployments
 - Create Services
 - Migrate data cleanly
 - Install a Kubernetes clusters
 - Create new types of resources

- Should an operator be deployed multiple times?
 - Yes
 - No

- When using Operators, there's no need to monitor services anymore
 - Yes
 - No



DEPLOY WORKFLOWS

Lesson 28: Deploy Workflows

Objectives

At the end of this lesson, you will understand:

- How to package Kubernetes manifests using Helm
- How to patch Kubernetes manifests using Kustomize
- How to maintain a history of your deployments
- Deploying with ArgoCD



What is Helm

<https://helm.sh>

- Kubernetes package manager: more than a collection of YAML
- Templating, Publishing, Metadata
- Application lifecycle: Service, Deployment, Secret, ConfigMap, ...



Helm Chart



A collection of Kubernetes manifests

- Templated with the Go template engine
- Metadata describe the application
- Optional dependencies with other charts



Chart Directory layout

- `Chart.yaml`: Chart metadata (application name, version, ...)
- `templates/*.yaml`: templated Kubernetes manifests
- `NOTES.txt`: Displayed at the end of installation (startup instructions, auto generated password, ...)
- `values.yaml`: default values for templated resources (image name/tag, application title/url)

```
myapp/
├── Chart.yaml
└── templates
    ├── deployment.yaml
    ├── _helpers.tpl
    ├── ingress.yaml
    ├── NOTES.txt
    ├── persistentvolumeclaim.yaml
    └── service.yaml

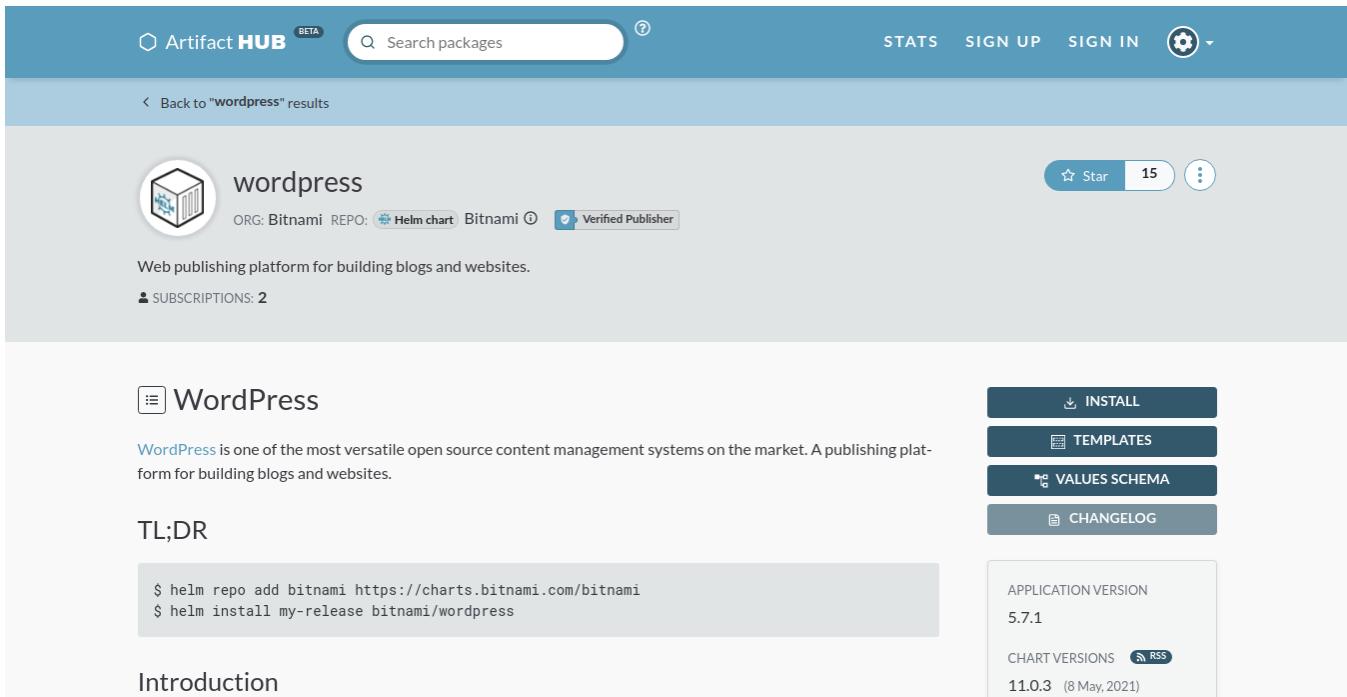
```

2 directory, 8 files



Artifact Hub

- A search engine for Hub repositories: <https://artifacthub.io>
- Anyone can add external repositories



The screenshot shows the Artifact Hub interface. At the top, there's a navigation bar with 'Artifact HUB' (BETA), a search bar, and links for 'STATS', 'SIGN UP', 'SIGN IN', and a gear icon. Below the header, a breadcrumb navigation says 'Back to "wordpress" results'. The main content area features a card for the 'wordpress' chart. It includes a thumbnail of a book, the name 'wordpress', the organization 'Bitnami', the repository 'Helm chart', and a 'Verified Publisher' badge. Below the card, it says 'Web publishing platform for building blogs and websites.' and 'SUBSCRIPTIONS: 2'. To the right of the card are four buttons: 'INSTALL', 'TEMPLATES', 'VALUES SCHEMA', and 'CHANGELOG'. On the left, there's a 'TL;DR' section with a code block showing Helm commands to add the chart and install it. On the right, there's a box for 'APPLICATION VERSION' (5.7.1) and 'CHART VERSIONS' (11.0.3, 8 May, 2021).



Demo: Wordpress Installation



- Install Helm
- Install a chart using Helm



Helm Usage

```
$ helm repo add c2c https://camptocamp.github.io/charts
$ helm search repo postgres
$ helm install app1 myapp/
$ helm list
$ helm template app1 myapp/
$ helm upgrade app1 myapp/
$ helm uninstall app1
```



Versioning strategy

- Like Docker images, the same chart should be used for dev and for prod
- Templating allows to make modifications to fit each environment
- We only need to change values from dev to prod

```
$ helm upgrade app1 myapp/ --set imageTag=latest
$ helm upgrade app1 helm-charts/myapp -f helm-values/myapp/dev/values.yaml
```



History and Rollback

- Helm keeps the history of deployments
- `helm upgrade` creates a new revision
- You can list revisions:

```
$ helm history <release name> --max=10
```

- you can rollback to a previous version:

```
$ helm rollback <release name> <revision>
```



Lab 28.1: Deploy a chart



Objective

- Deploy a complete application

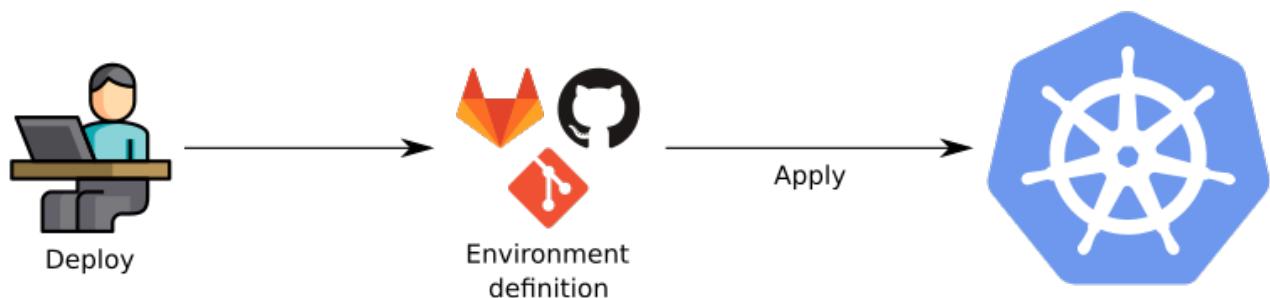
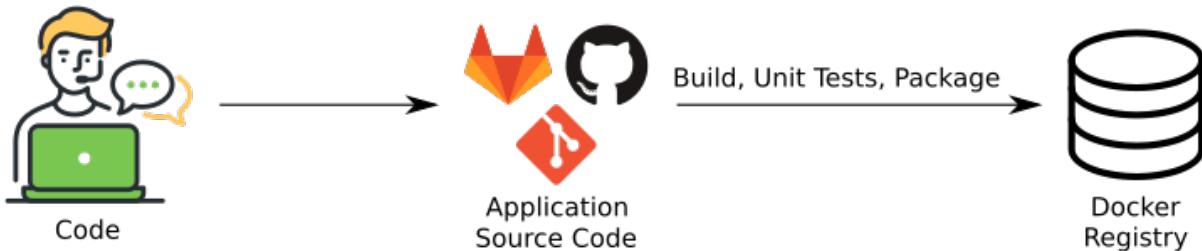
Steps

- Search for an interesting chart with the `helm search repo` command
- Deploy this application
- Find the URL of the new application and use it



GitOps

 **GitOps** Everything needed to deploy the environment is versioned in Git



Helmfile

- Wrapper around Helm
- Declarative `helmfile.yaml`
- Pull multiple versioned charts together
- GitOps with CI/CD



ArgoCD



Log in on <https://argocd.apps.k8s-training.campto.camp/>



Lab 28.2: Use ArgoCD GitOps



Objective

- Deploy using GitOps on ArgoCD

Steps

- Clone <https://github.com/camptocamp/containers-course-argocd>
- Fork the `template` branch as branch named after your login:

```
$ git checkout template  
$ git checkout -b raphink
```

- Create a Helm chart in the top directory

```
$ helm create .
```

- Edit the Helm chart to deploy your application
- Commit and push
- Visualize result on ArgoCD and K9s



Checkpoint: Deploy Workflows

- Can we run multiple instances of the same chart?
 - Yes
 - No
- Can you create a chart for your custom application?
 - Yes
 - No
- Does helm allow to version LoadBalancer definitions?
 - Yes
 - No
- Does helm provide an idempotent command to install or update release?
 - Yes
 - No



SECOND DAY OPERATIONS

Lesson 29: Second Day Operations

Objectives

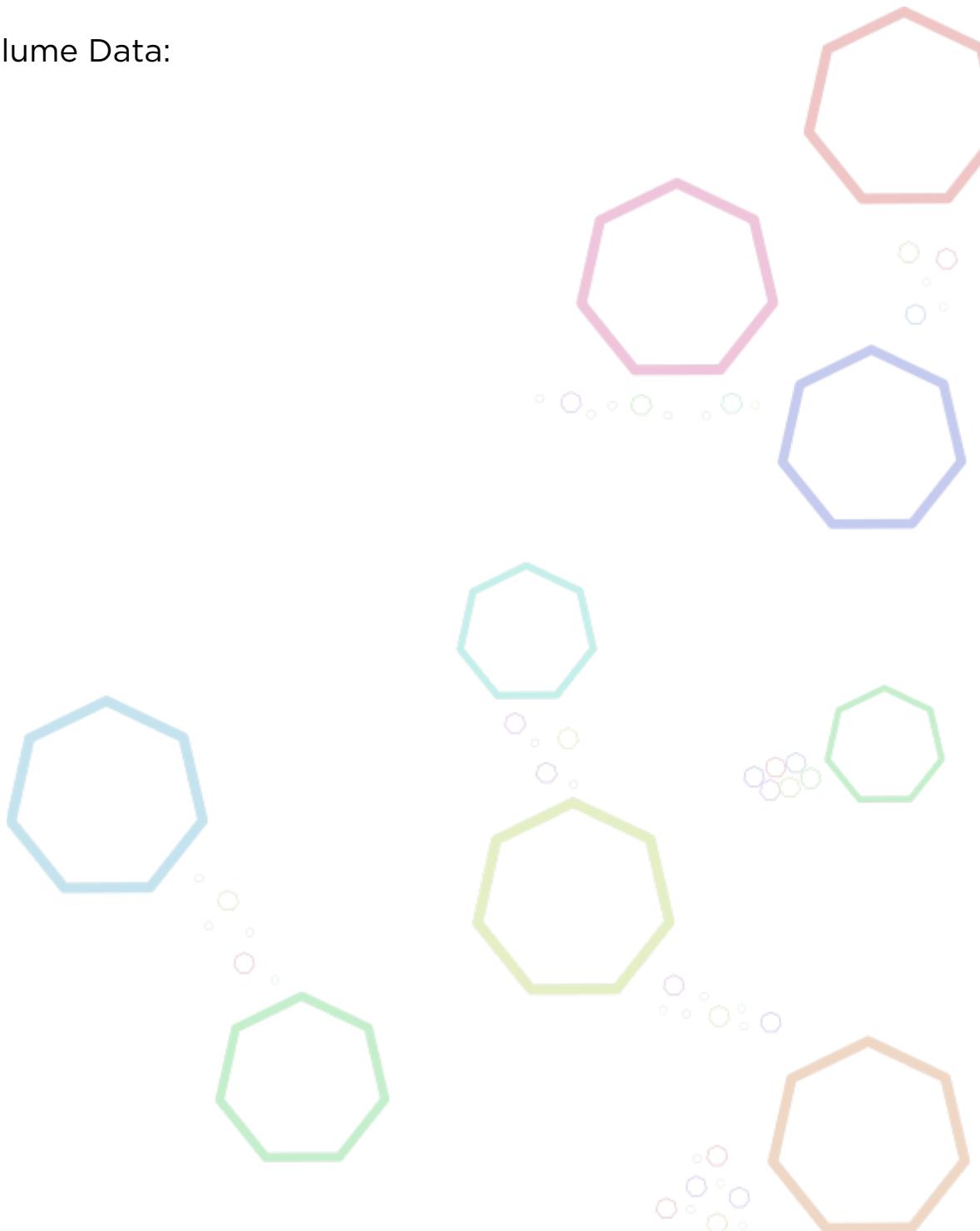
At the end of this lesson, you will know about:

- Kubernetes backups
- How to organize deployment
- How to manage severals env
- Keep efficient development
- Observability



Backups

- Generally not necessary
- Backup volume or backup NAS
- Backup etcd objects:
 - Velero
- Backup Volume Data:
 - Bivac
 - K8up



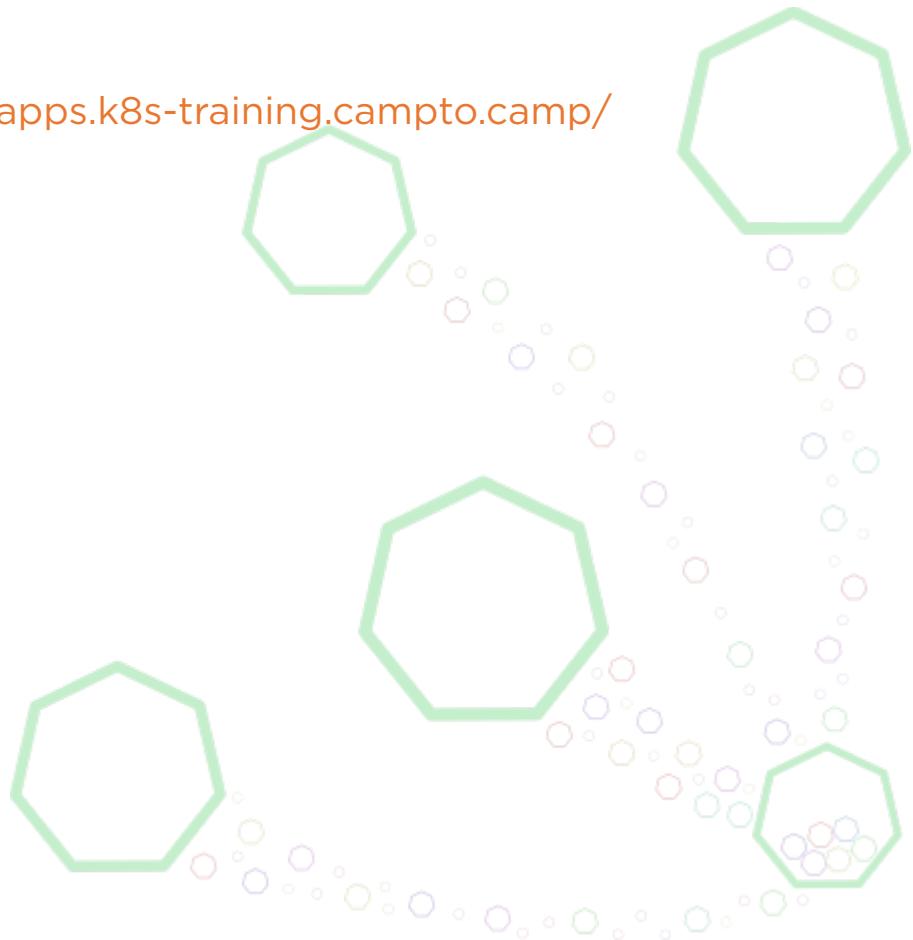
Prometheus

- Made for Container monitoring
- Inspired by Google Borgmon (tool to monitor Borg)
- Second CNCF project (after Kubernetes)



- Time series
- Labels
- Federation

Demo: <https://prometheus.apps.k8s-training.campto.camp/>

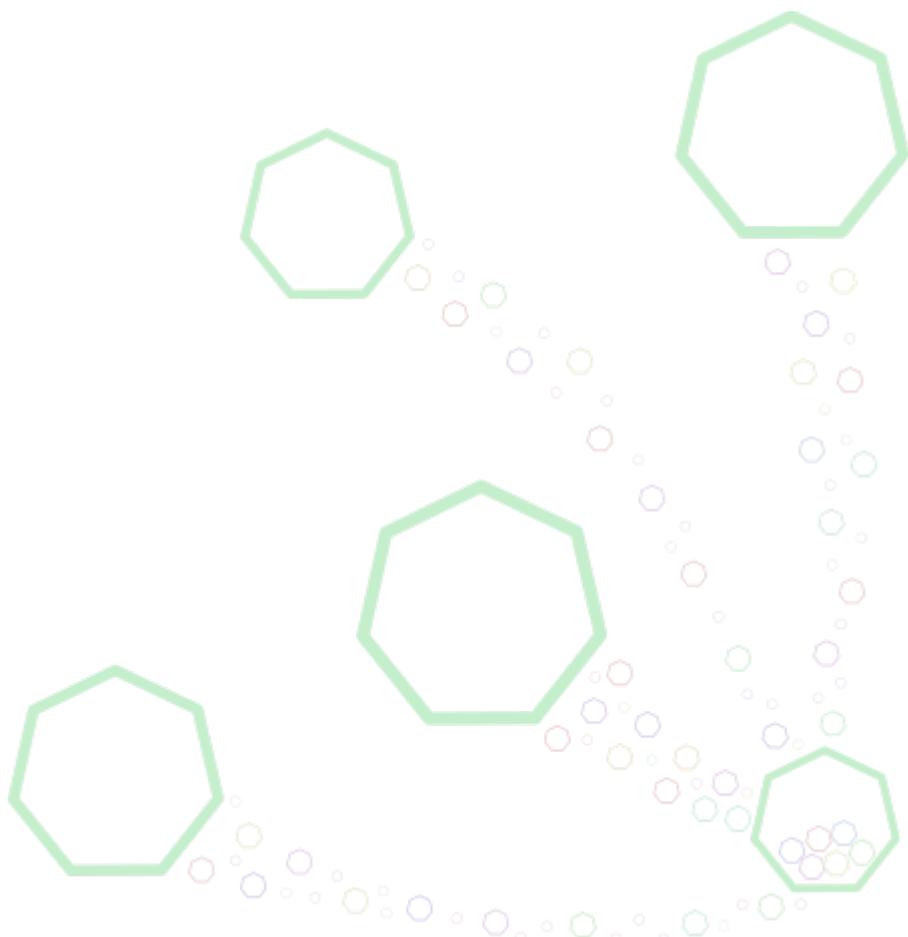


Prometheus Operator



CRDs:

- *ServiceMonitor*: Add target on *Endpoints*
- *PodMonitor*: Add target on *Pod*
- *PrometheusRule*
 - Alert
 - Recording rules

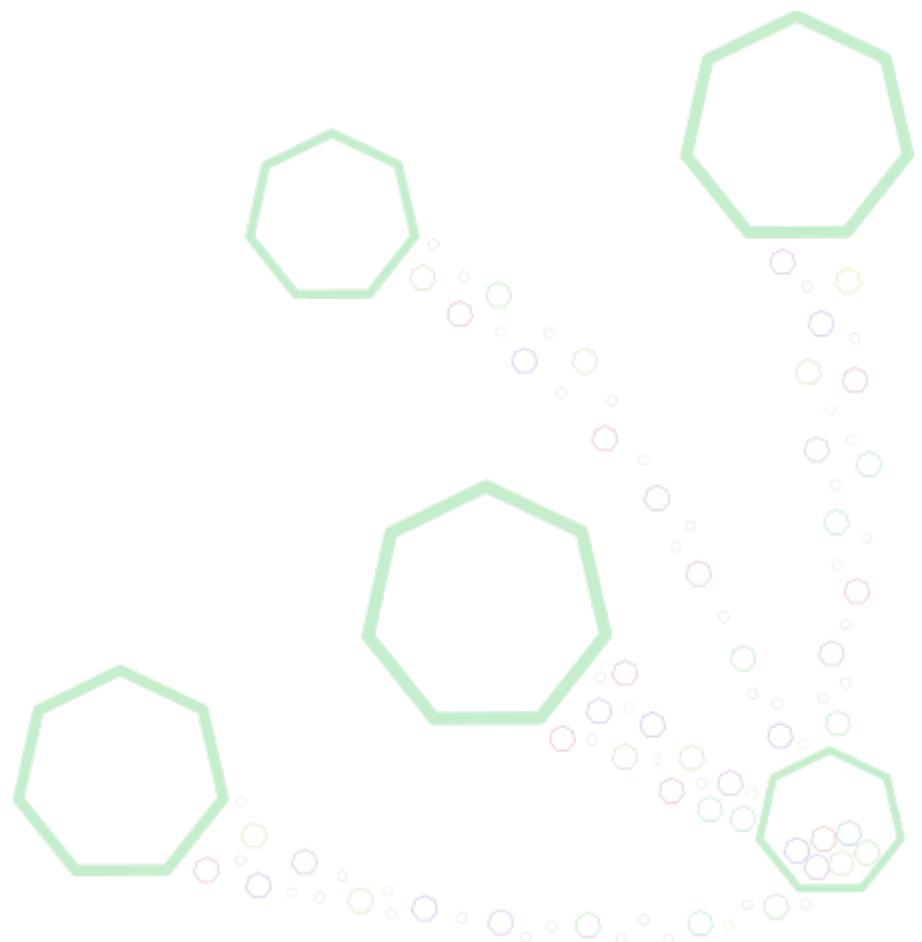


Grafana

- Web interface for graphs
- Supports multiple data sources (including Prometheus)



Demo: <https://grafana.apps.k8s-training.campto.camp/>



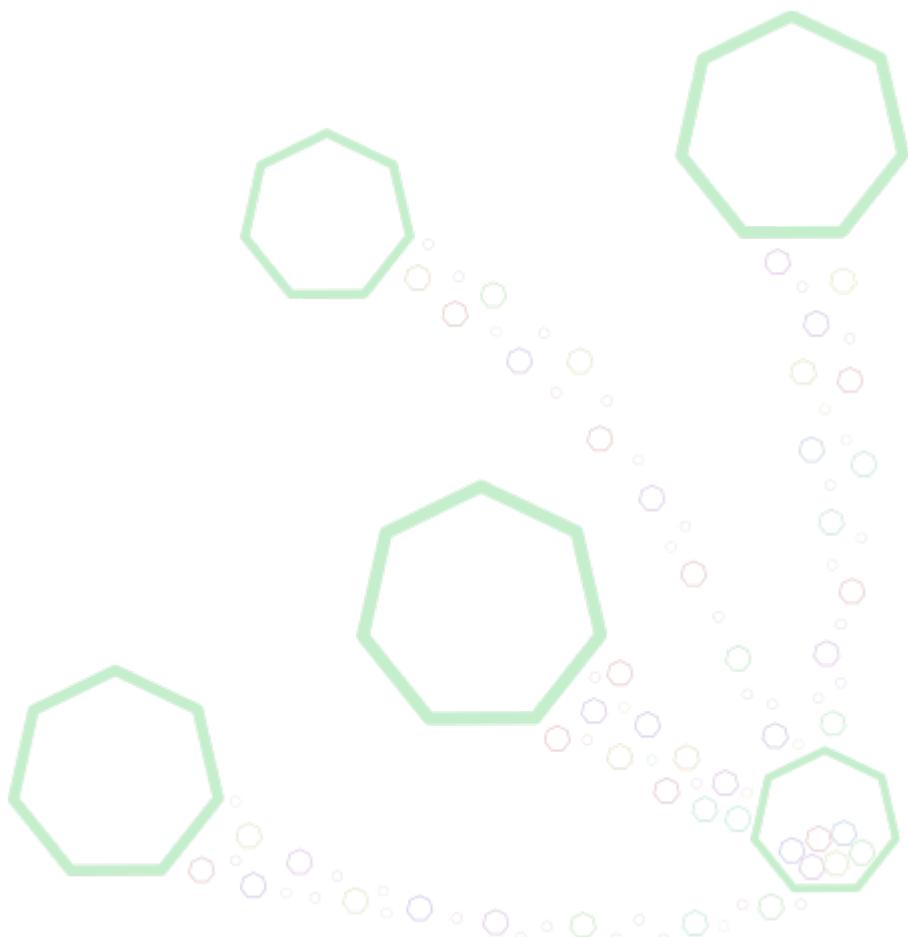
Alert Manager

Alerts defined in Prometheus (along with recording rules):

<https://prometheus.apps.k8s-training.campto.camp/rules>

- Can be linked to Data source
- Can be silenced

Demo: <https://alertmanager.apps.k8s-training.campto.camp/>



GitOps

What is GitOps:

- Source of truth is git repository
- Everything is committed somewhere: image tag, replica count, secrets to use
- Any modification requires a git commit
- Manual modifications are still possible but will be reverted



Git is not mandatory, any decent version control system can be used.

Why GitOps:

- Be able to rebuild everything in case of disaster recovery.
- Have the history of application deployment.
- Be able to automate deployment.

GitOps — Build artifacts

Use a Git repository for:

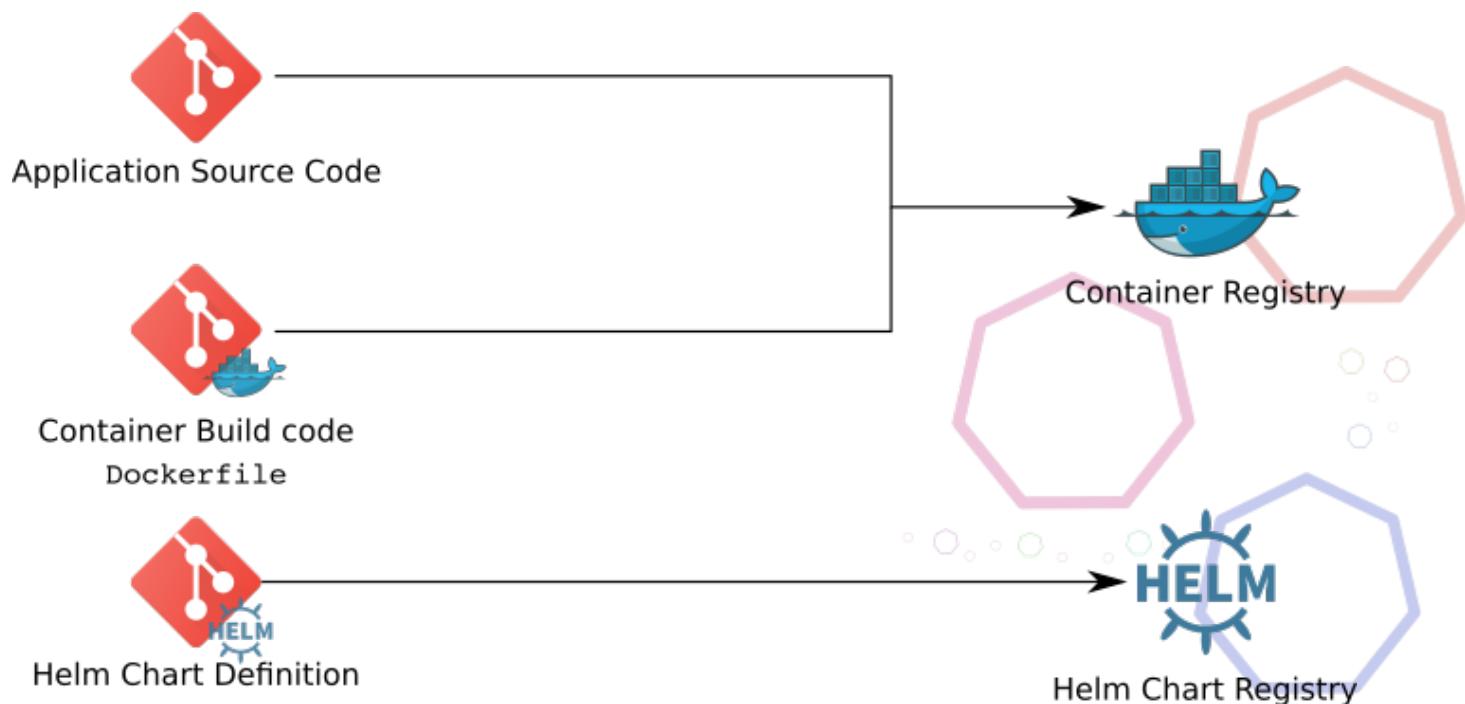
- Application source code
- Container Creation: `Dockerfile`
- Helm Chart

You can commit everything in the same repository or use a dedicated repository.



GitOps – Build artifacts

- Dockerfile and source code will be used to create one artifact: the Container Image.
- Helm Chart definition will be used to create another artifact: the Helm Chart



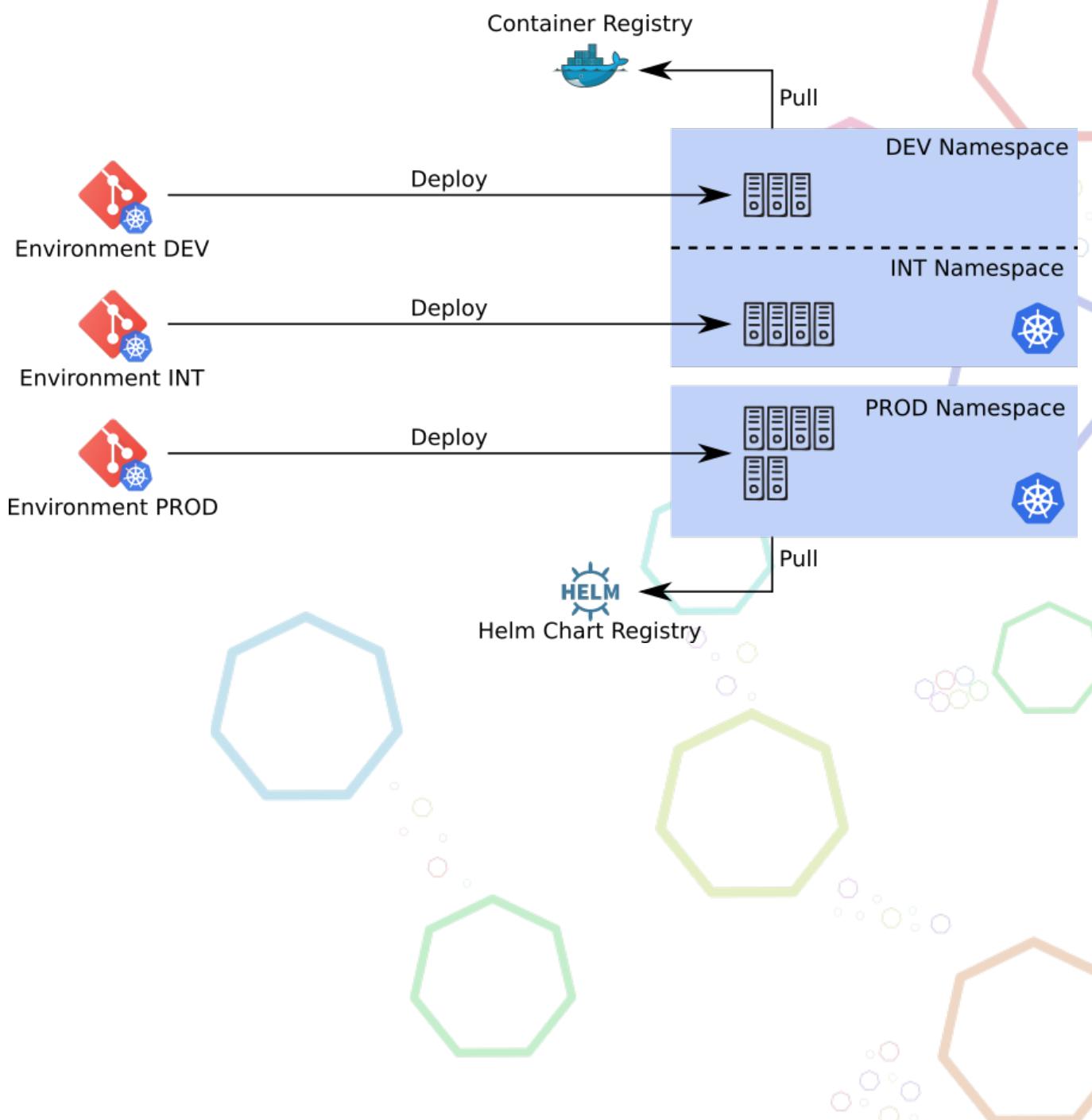
So using one repository per artifact may be better.

What does a git tag means if source code and Chart definition are in the same repository?

GitOps - Deployments

Definition of applications to deploy in a specific environment need a dedicate repository.

- One git repository per environment: More easy to maintain specific values per env
- One branch per environment: Synchronization of env can use standard git merge strategy



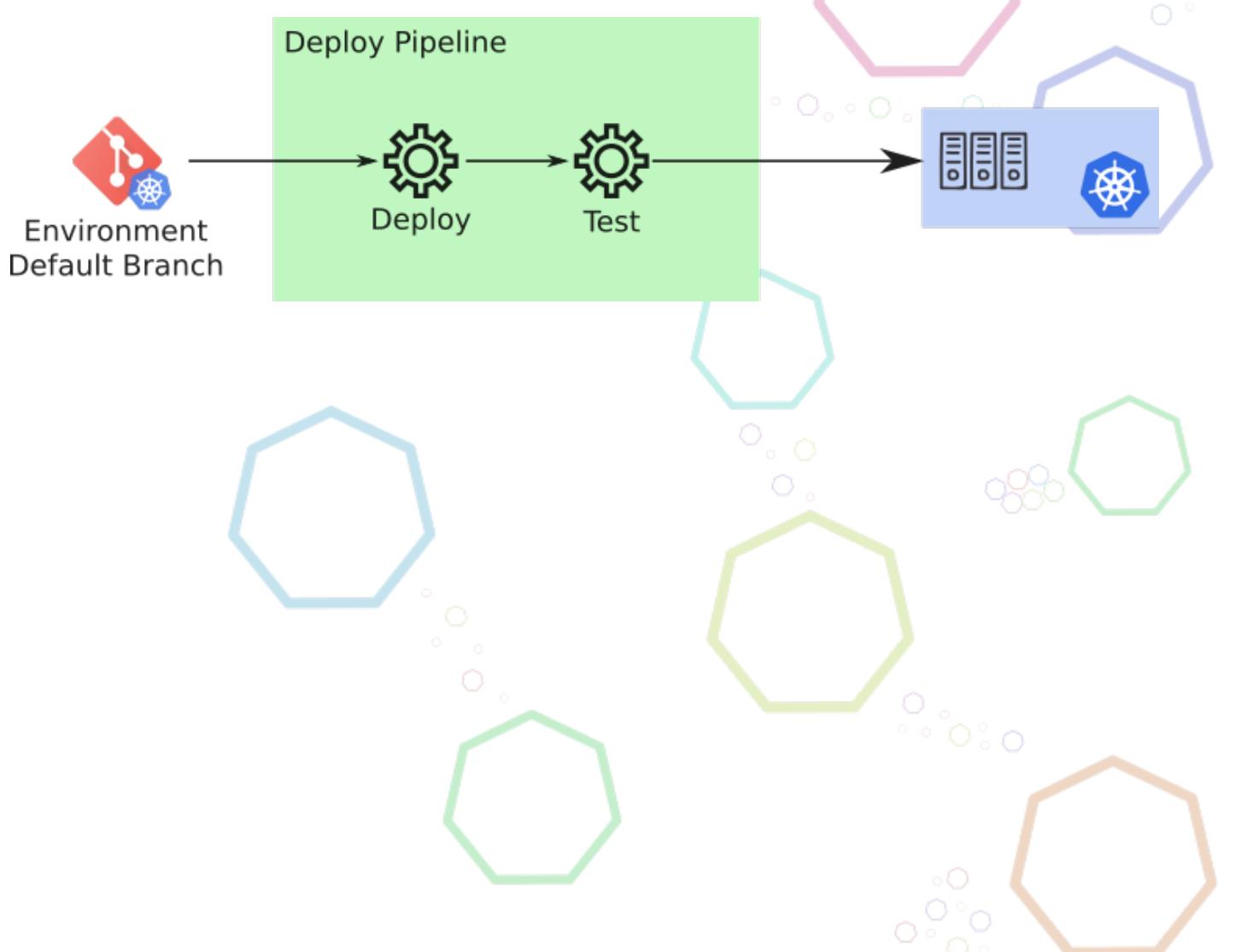
Automated Deployment

Motivation:

- Improve Quality of deployment by running automatic tests
- Reduce repetitive work to deploy last version of application

Automated Deployment:

- Deployment is scripted and automated
- Used to deploy from scratch and for continuous update
- New commit on default branch are deployed automatically
- Run test, benchmark after deployment



Automated Deployment

Observability

Deploy pipeline can export metrics about deployment status:

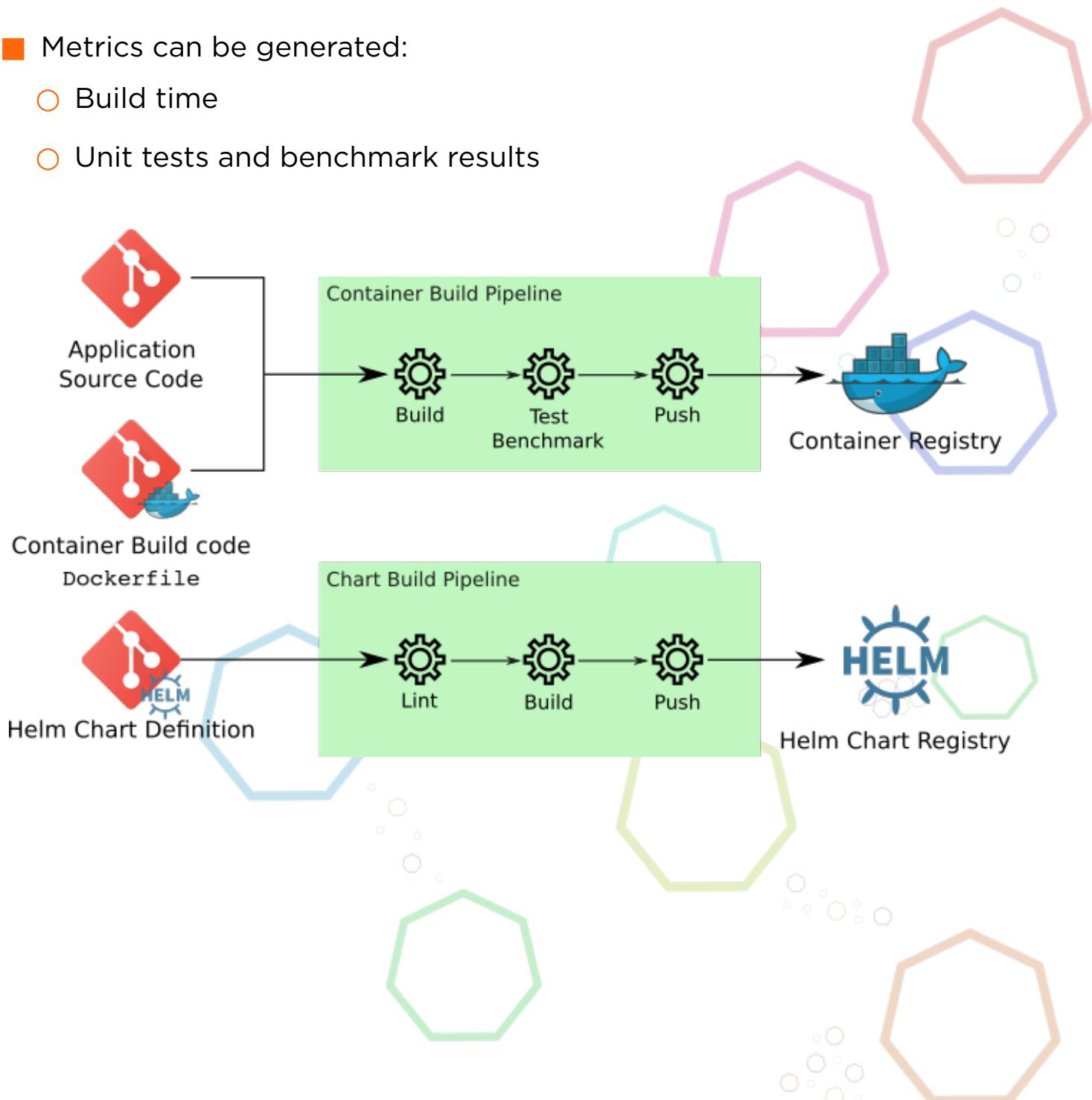
- Add [dashboard annotations](#) with the [Grafana API](#).
- Annotation contains a text field and a list of tags.
- Can be a range.
- Notifications can also be sent to users.

Deploy from workstation?

- By-pass automation, can deploy uncommitted modifications
- Need secrets on workstation
- Useful in case of disaster recovery: reduce dependencies
- Be able to deploy without Git forge or pipeline runner/worker
- Pipeline definition and deploy script need to be versioned and tested

Automatic Build

- Automatic build of artifacts: Container image and Helm chart
- Triggered by commit:
 - At least on default branch
 - Maybe on feature branch or merge requests
- Metrics can be generated:
 - Build time
 - Unit tests and benchmark results



Automatic Build - Artifact tag

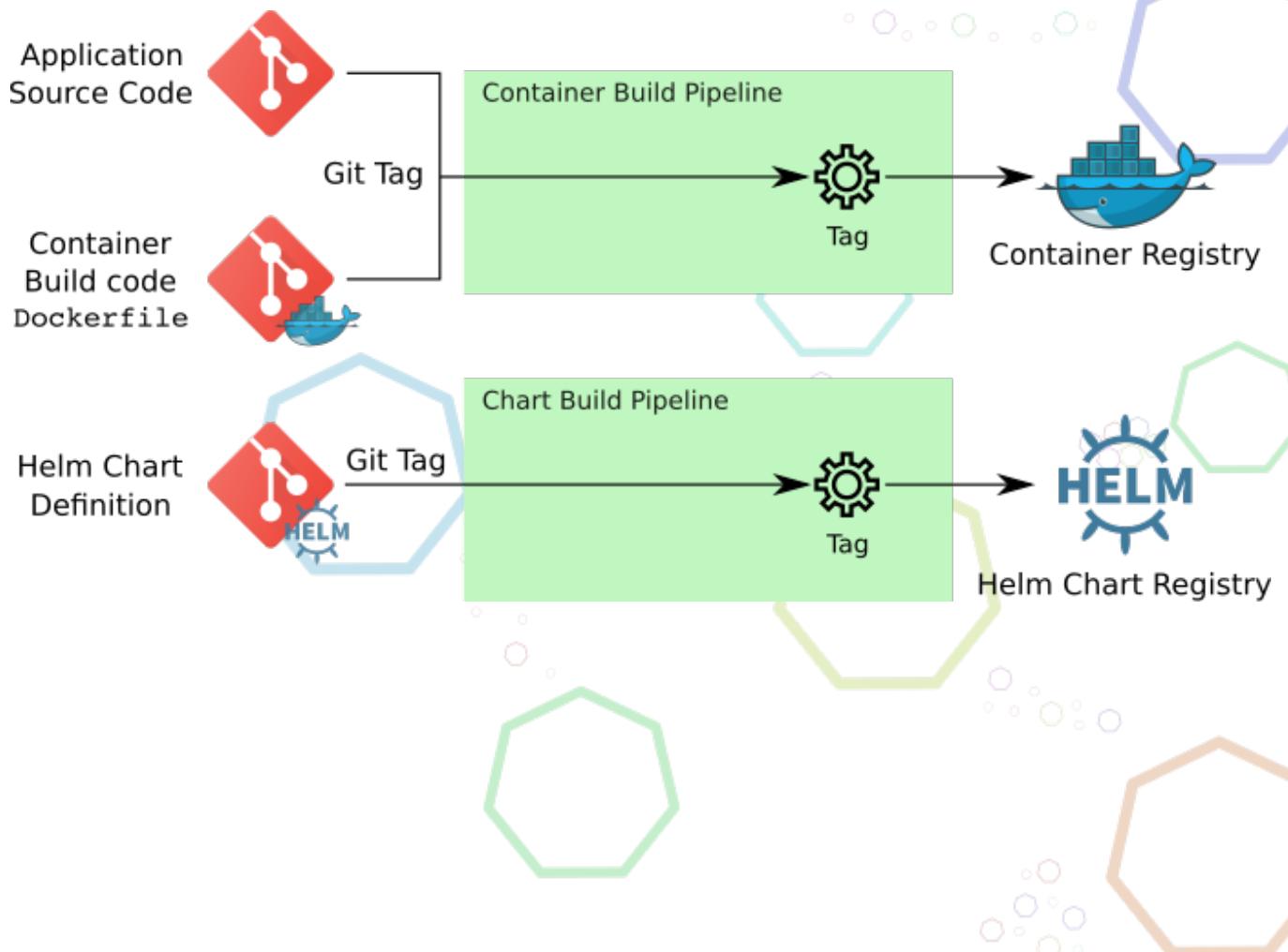
Artifact tags are used to identify source of the build:

- `latest` : last build on default branch
- `<branch>-<commit ref>` : build of specific commit on default or feature branches. Ex: `add_auth-6dd68f7` or `master-b9056df`
- `<git tag>` : Build corresponding to the git tag



In the Helm Chart metadata there is a `version` field. It's better to not use this field and compute version based on branches, tag and commit ref.

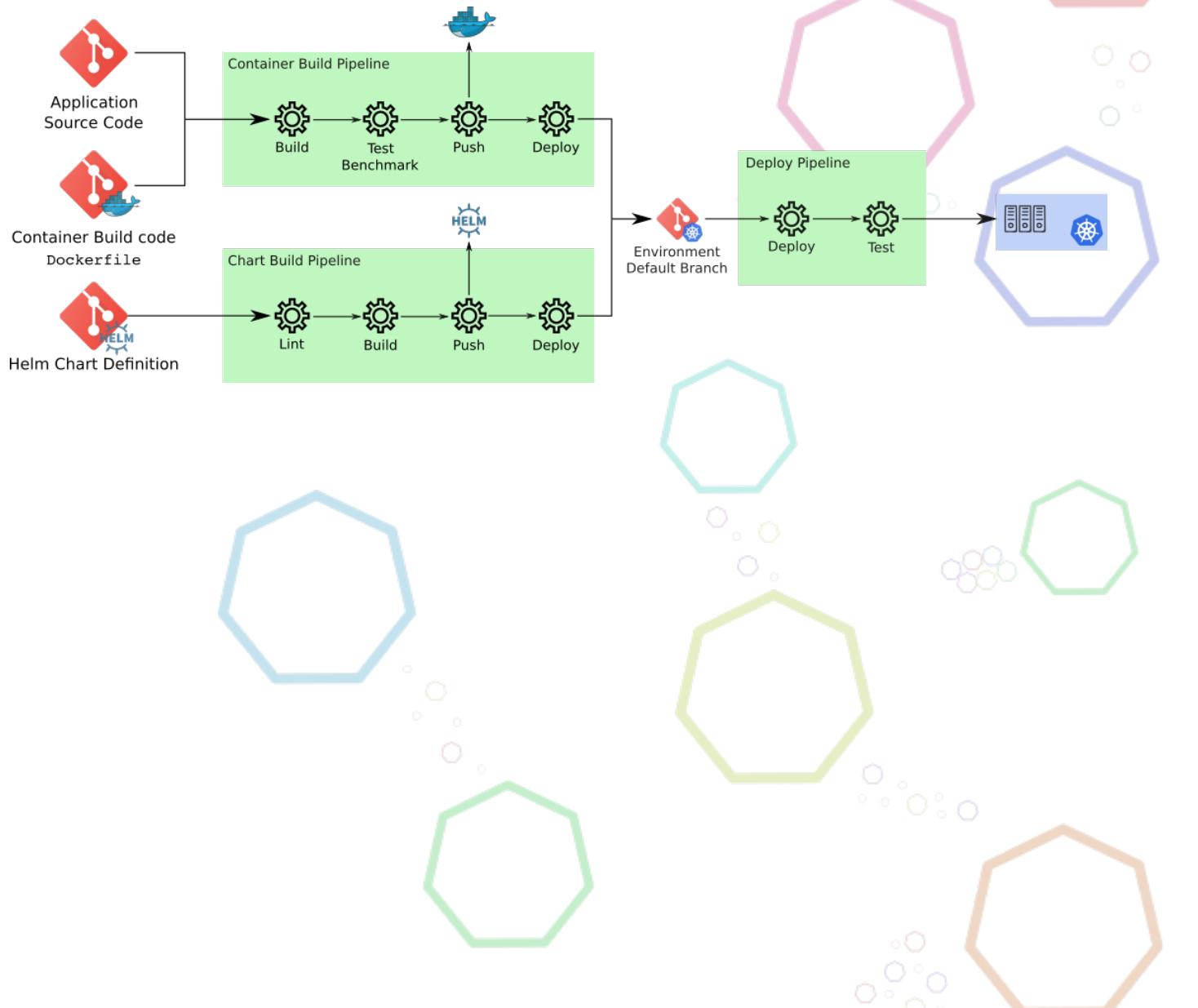
Git tags are just alias of already existing commit, when a git tag is pushed there is no need to rebuild.



Continuous Integration

New artifacts are automatically deployed in a test environment:

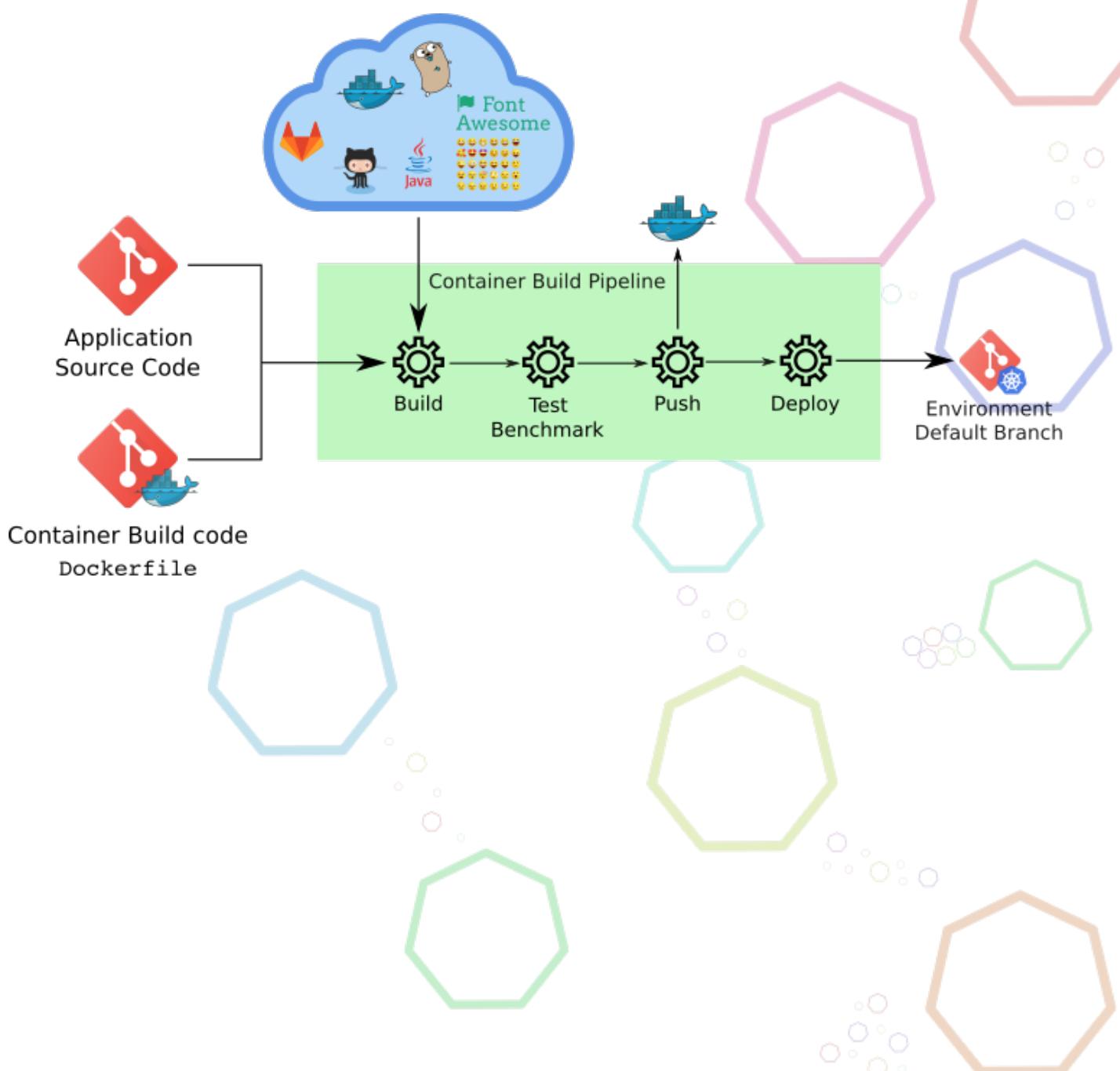
- Dev environment for commit on default branch
- Temporary environment for feature branch or merge requests
- Git tag on source code can trigger deployment in the Int environment
- Run integration tests
- Build pipeline will commit on environment repository



Management of external dependencies

Every software has dependencies:

- Source code maintained by community
- Upstream docker image used as base image
- Language specific libraries
- Images, Sounds, Icons, Fonts



Management of external dependencies

Consolidation of dependencies to ensure reproducibility of the build:

- Private container registry
- Nexus Repository

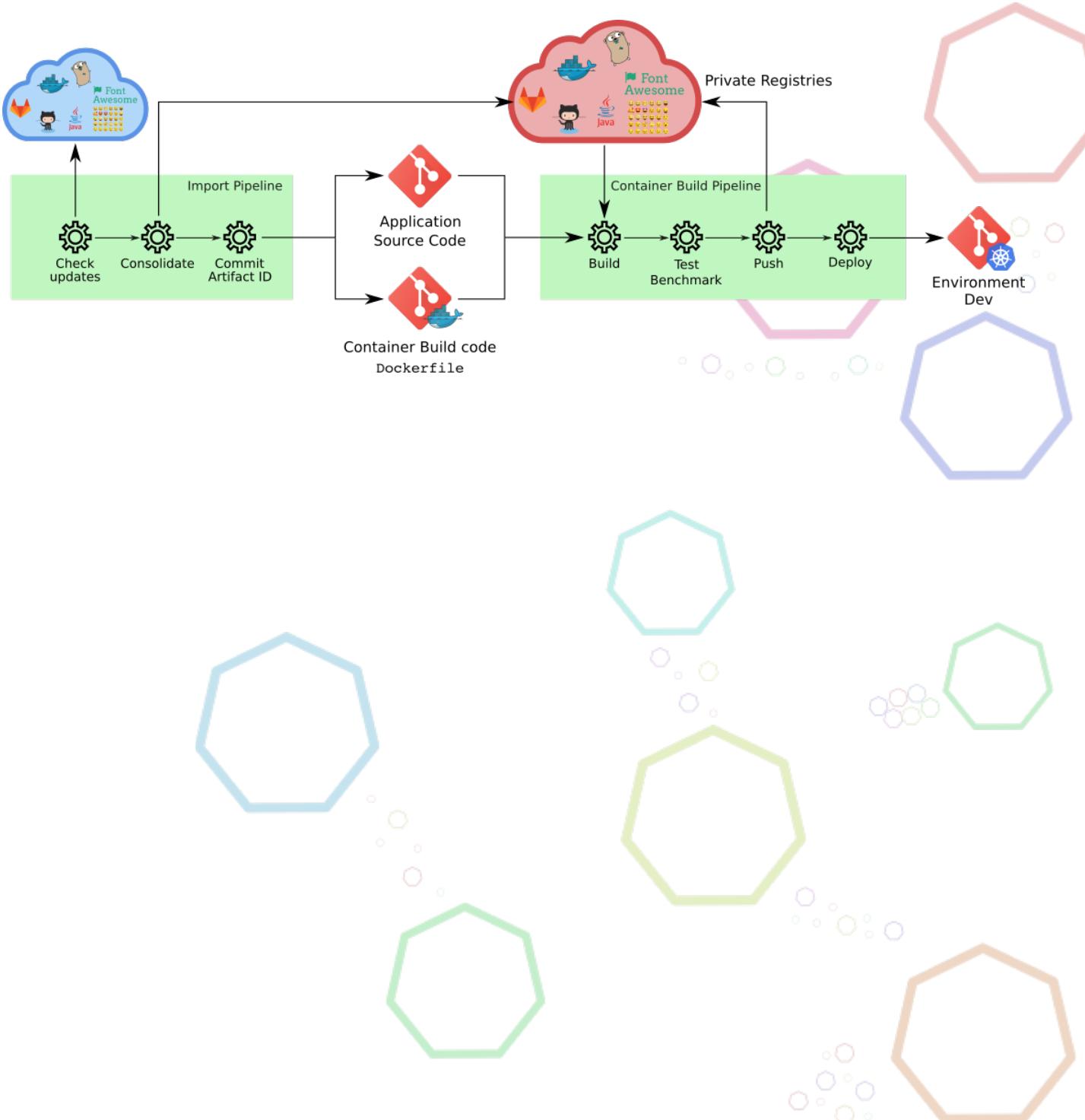
Some dependencies updates include security patches or bug fix:

- Container used as base image
- Language libraries



Automatic include of upstream patches

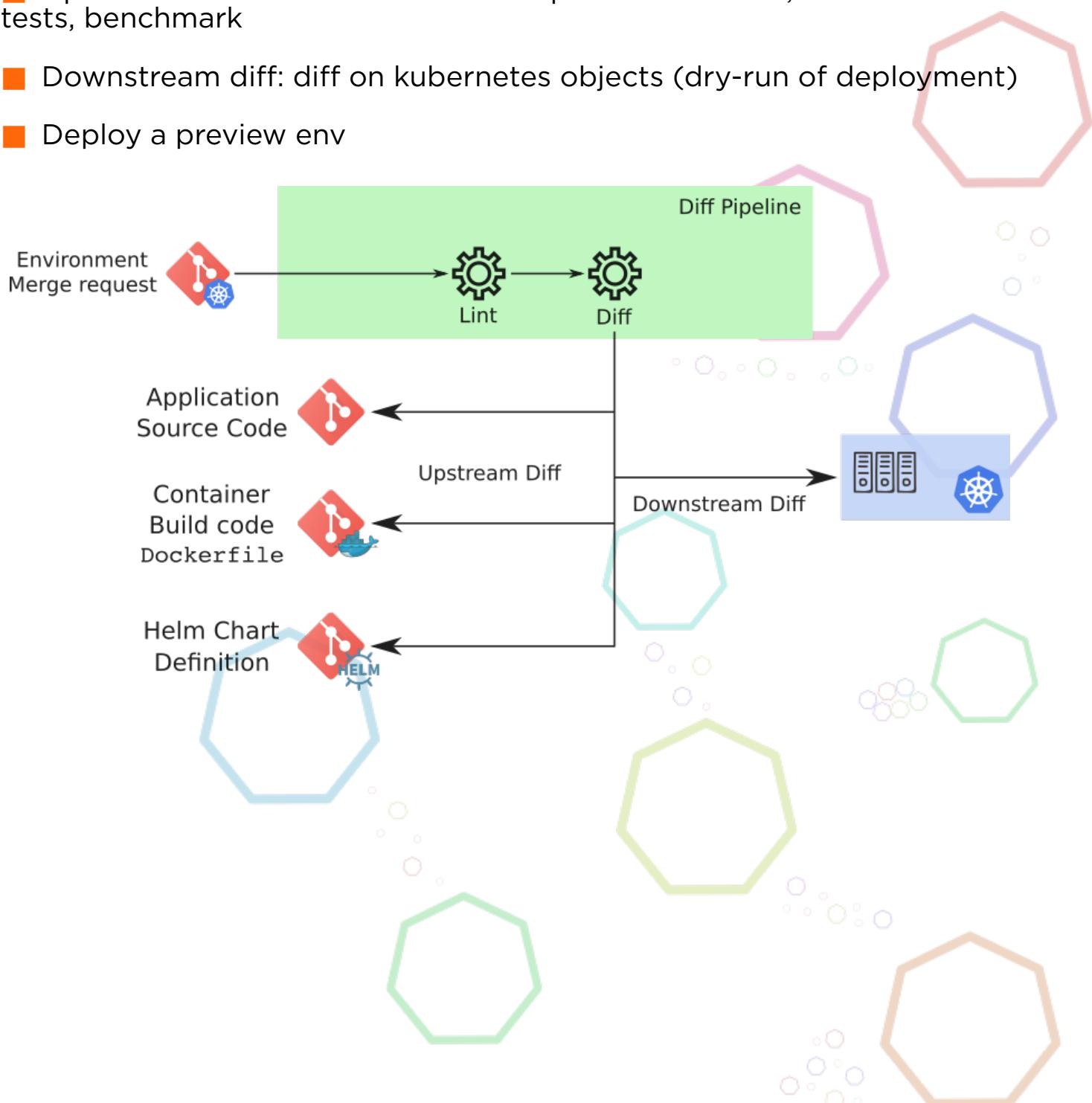
- Monitor upstream images and artifacts every night
- Commit upstream artifacts ID in source repository
- Trigger rebuild
- Deploy on dev environment



Full Recursive Deep Diff

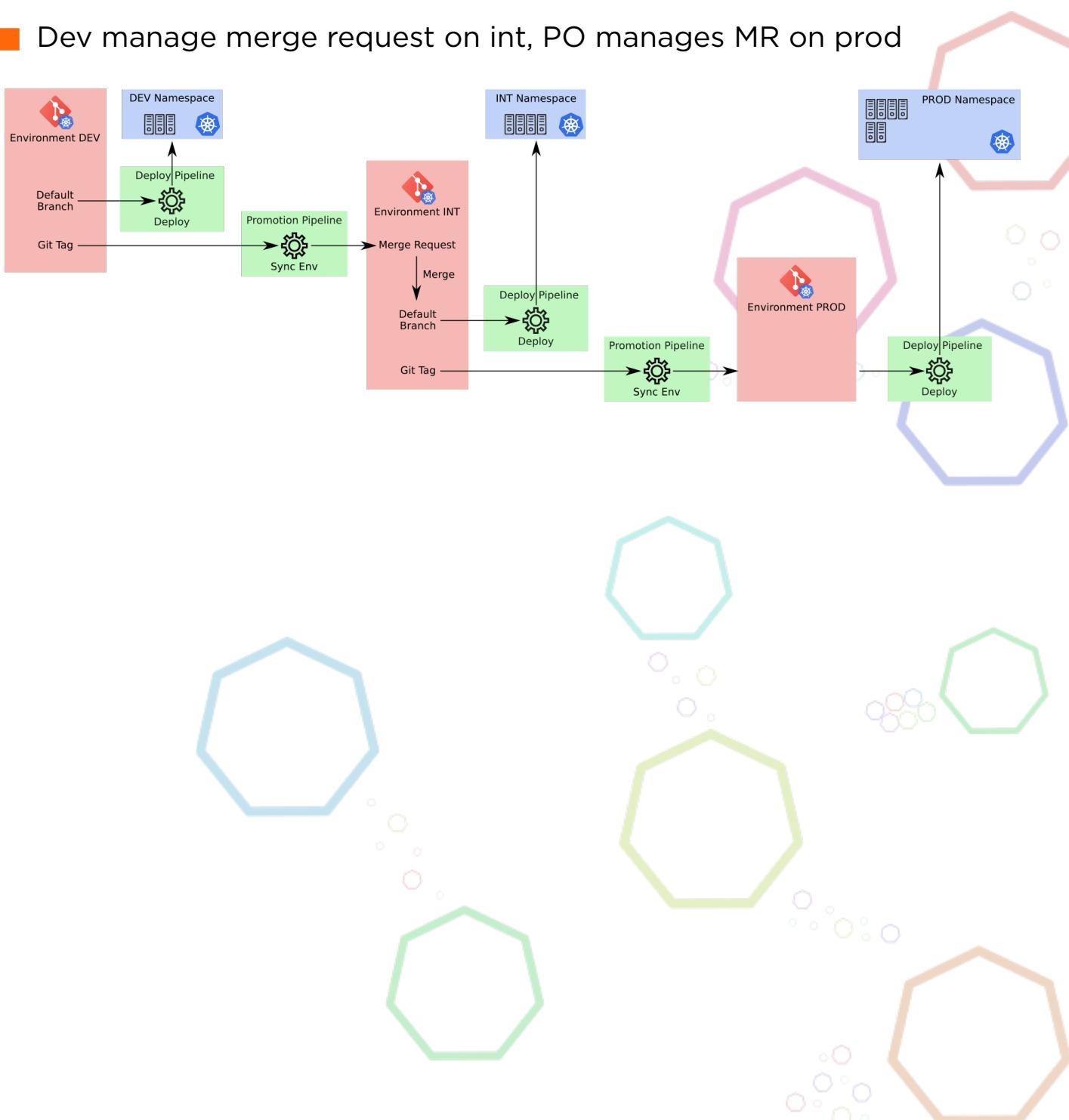
In the environment repository, we need a diff before merging a merge request:

- Git diff on merge requests is not meaningful: only change of artifact version (2.0.5 -> 2.1.0)
- Upstream diff: diff in source code repo or helm chart, result of unit tests, benchmark
- Downstream diff: diff on kubernetes objects (dry-run of deployment)
- Deploy a preview env



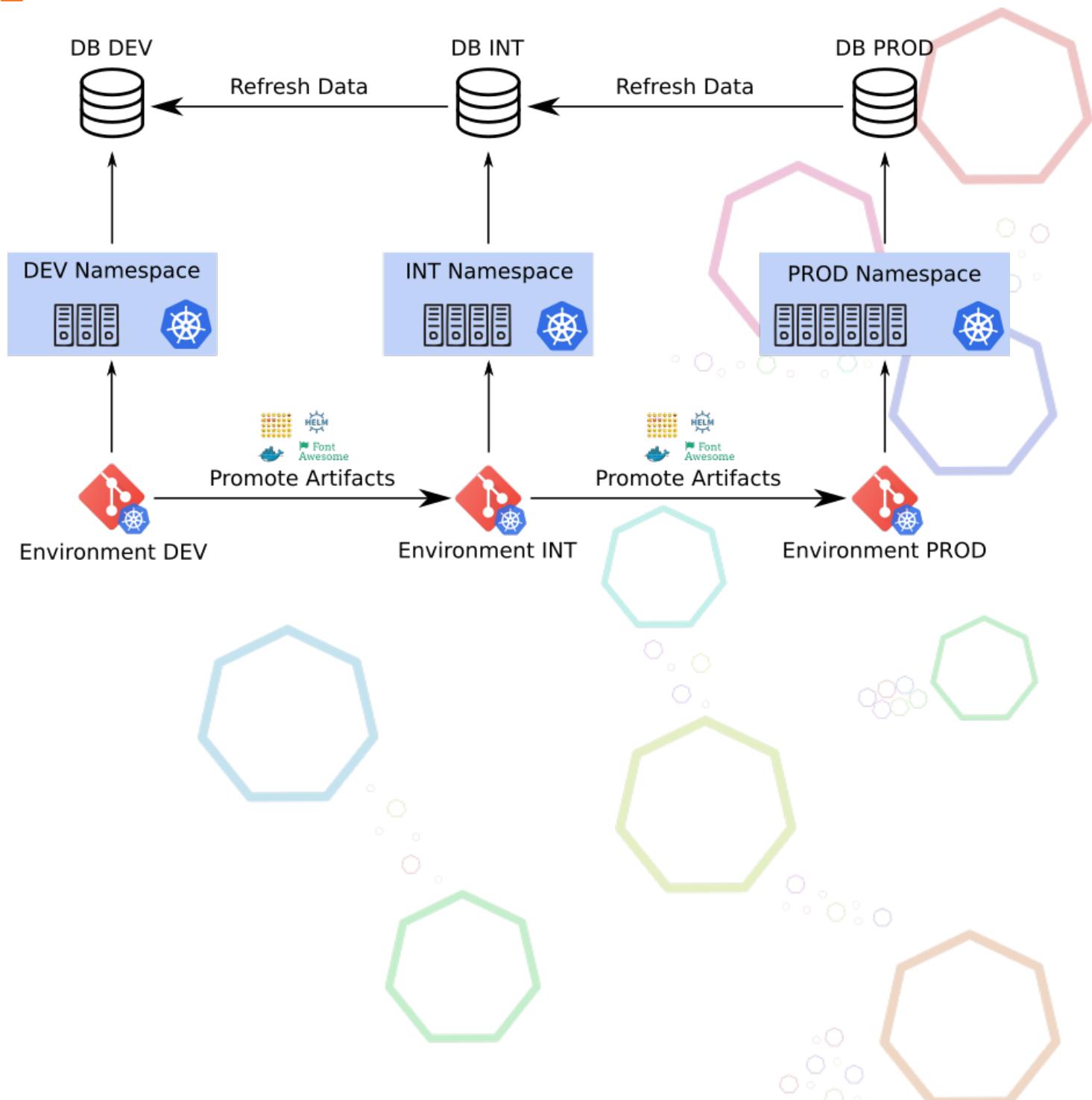
Promotion & Automation

- Docker image promotion, no rebuild for prod, only build for dev
- Diff between env: dev / int, int / prod
- Git tag create merge request in the next env: dev -> int -> prod
- Automatic deploy for dev, manual merge for prod
- Dev manage merge request on int, PO manages MR on prod



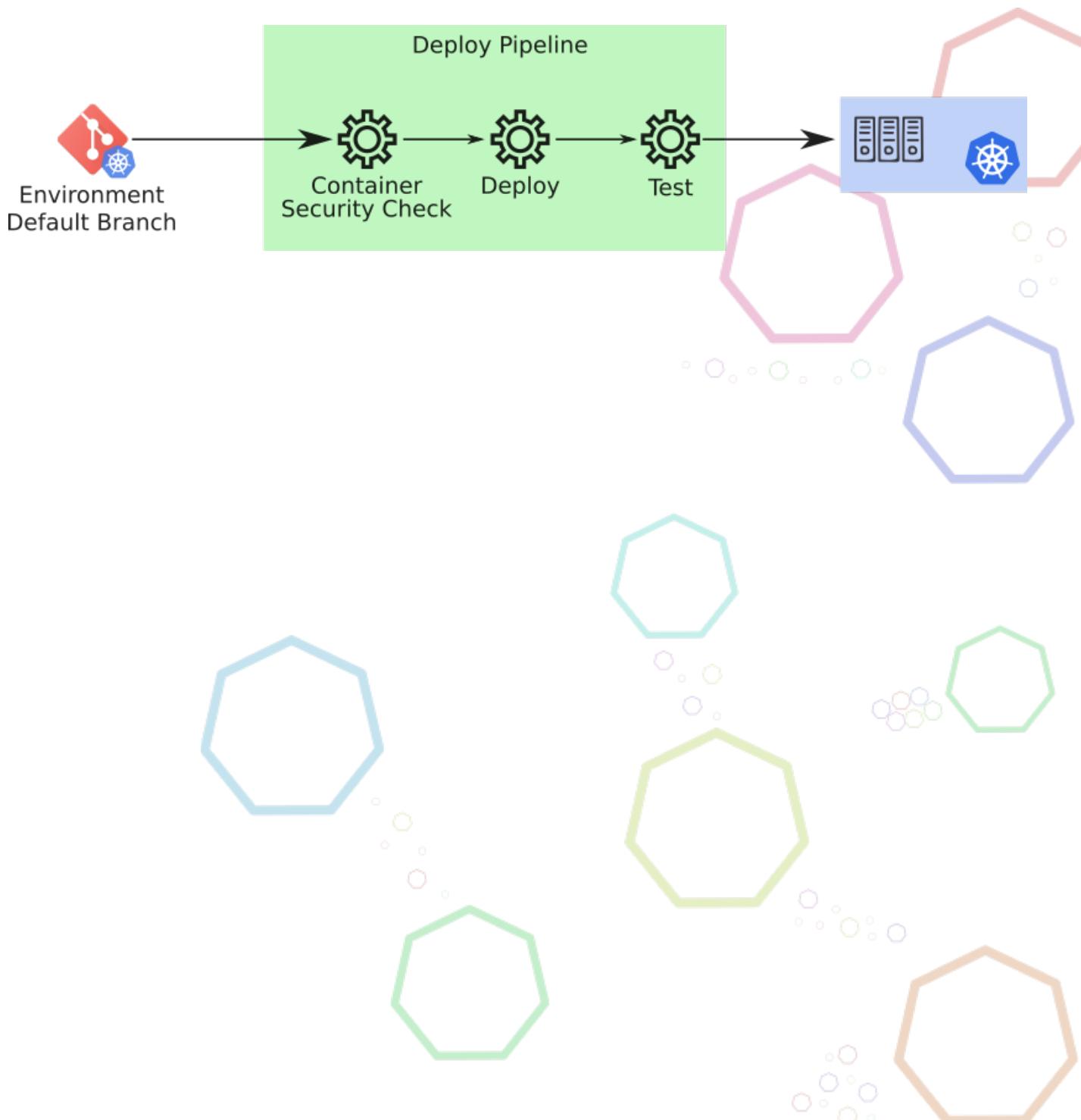
Data management

- Some Data should be refresh from prod:
 - products, core business components
- Some data follow code: static files, asset
- Test data



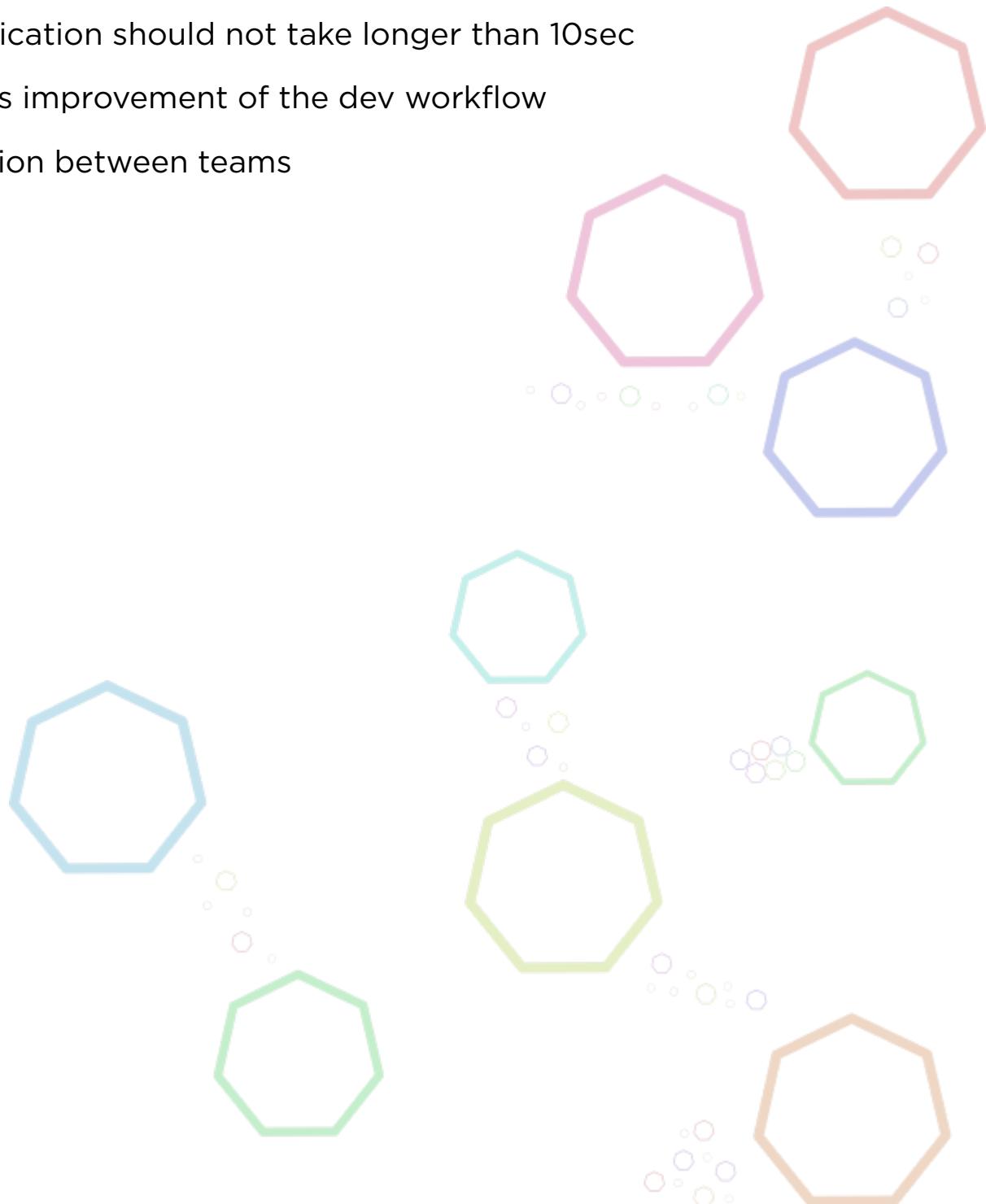
Security Scan

- Every image builded is scanned against vulnerability
- Block deployment of images that contains vulnerability
- Static scan, dynamic scan
- Alerts based on reports



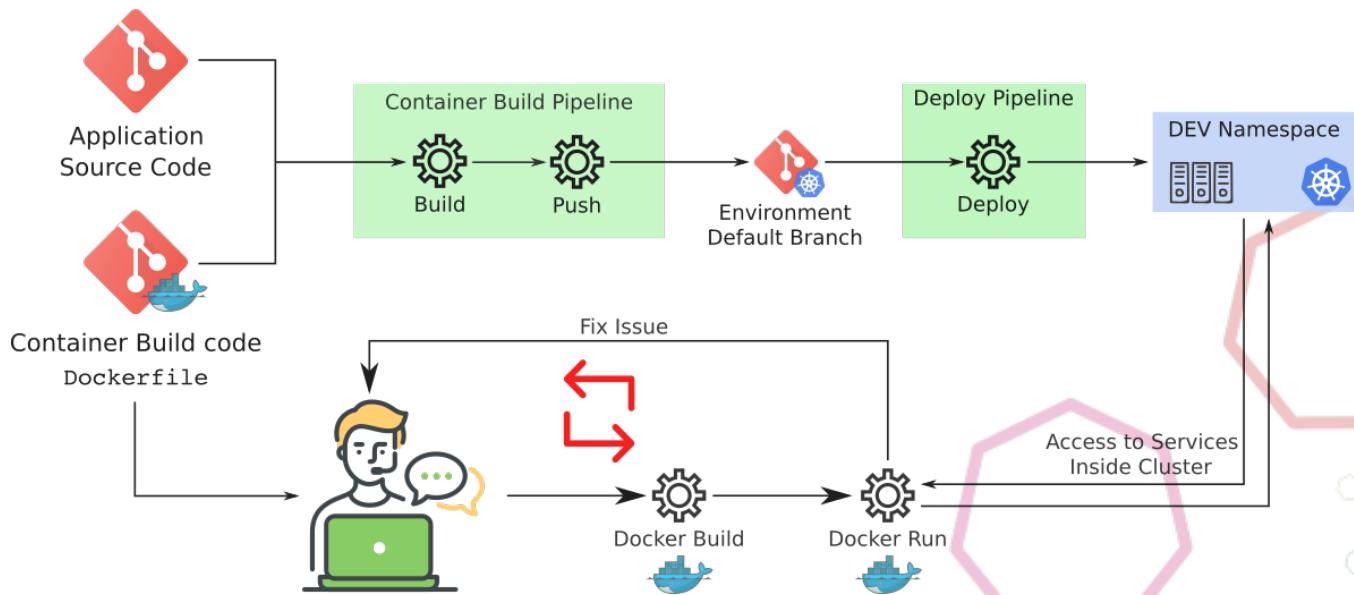
Take care of developpers

- Automation can slow down development
- Dev want to test before commit
- Test is done by deploying
- Deploy local manifests
- Test modification should not take longer than 10sec
- Continuous improvement of the dev workflow
- Collaboration between teams



Take care of developpers

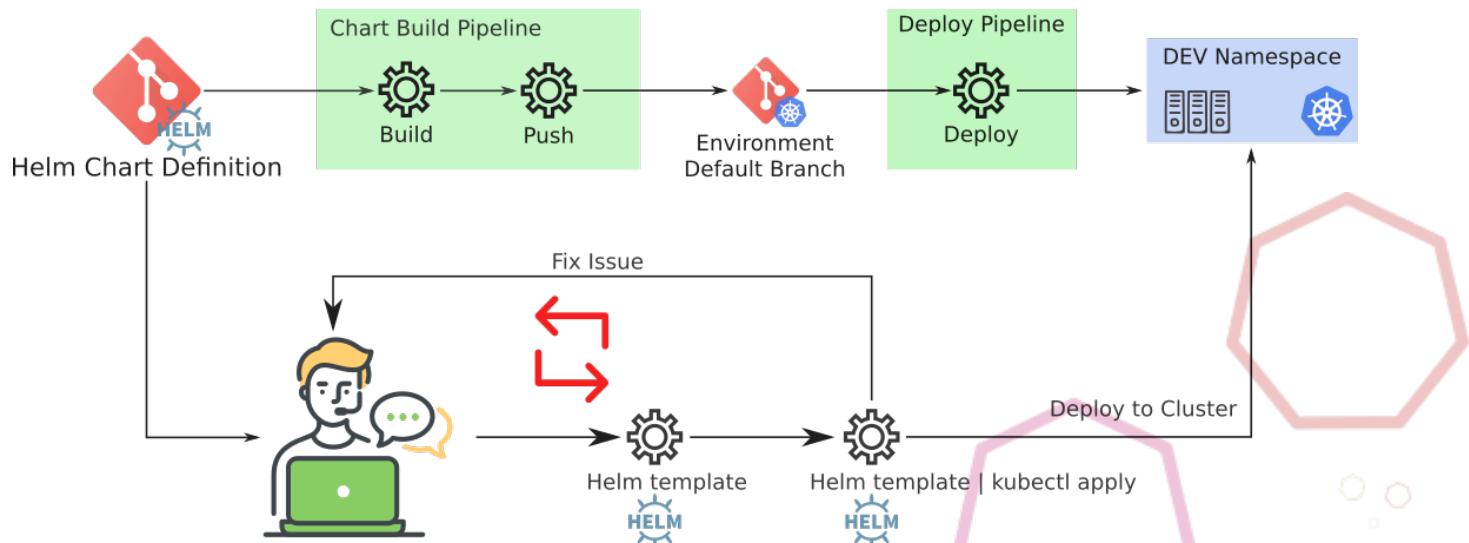
Working On Container Image



- Communication between local container and resource in cluster is still a challenge.
- Final step is to commit.

Take care of developpers

Working On Helm Chart



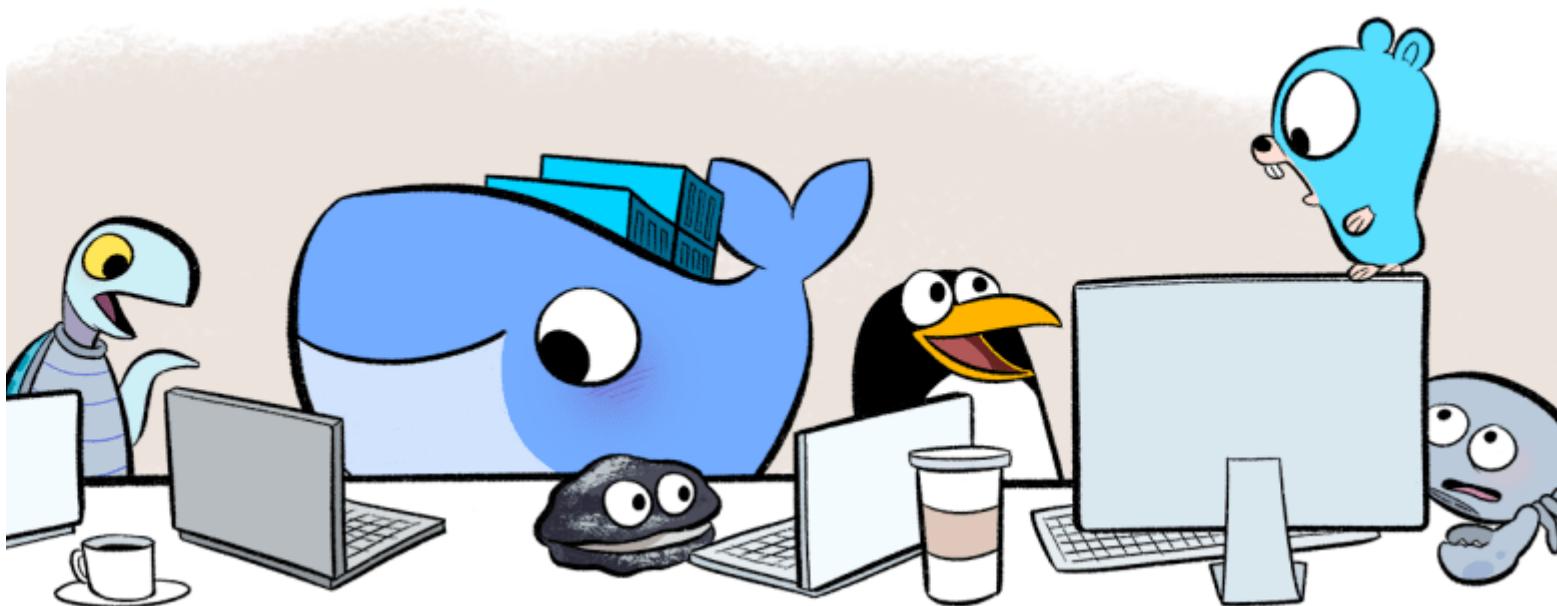
- Need cleanup before and after deployment.
- Final step is to commit.

COURSE CONCLUSION

Course Summary

During this class, we:

- Downloaded, ran, built & orchestrated Docker images
- Created, updated & destroyed Kubernetes objects
- Inspected the Kubernetes API, dashboard and `kubectl` CLI
- Explored Kubernetes architecture and network principles



WWW.CAMPTOCAMP.COM

camp

Open Source specialist, Camptocamp consists of three divisions: **GEOSPATIAL SOLUTIONS**, **BUSINESS SOLUTIONS** and **INFRASTRUCTURE SOLUTIONS**. Our professional, innovative and responsive services help you implement your most ambitious projects.

SWITZERLAND Quartier de l'Innovation EPFL / PSE-A / CH-1015 Lausanne / Tel +41 21 619 10 10
FRANCE Savoie Technolac / BP 50352 / F-73372 Le Bourget-du-Lac / Tel +33 4 58 48 20 20
GERMANY Linuxland GmbH / Thomas-Dehler-Str. 9 / D-81737 München / Tel +49 89 99 34 14 40

Follow us

