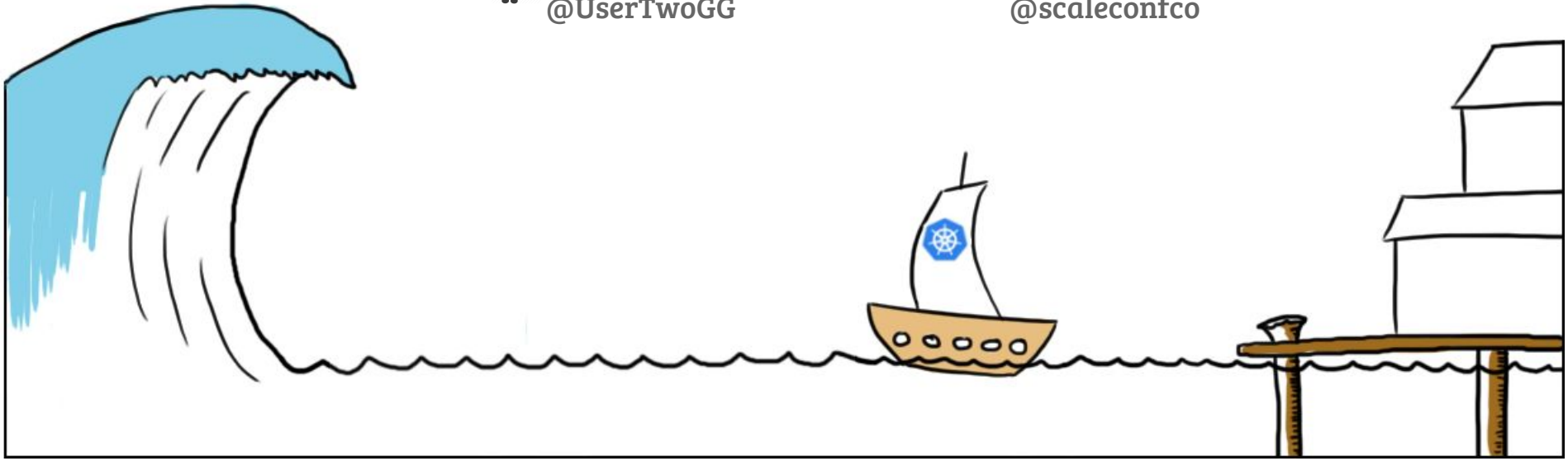
Kubernetes

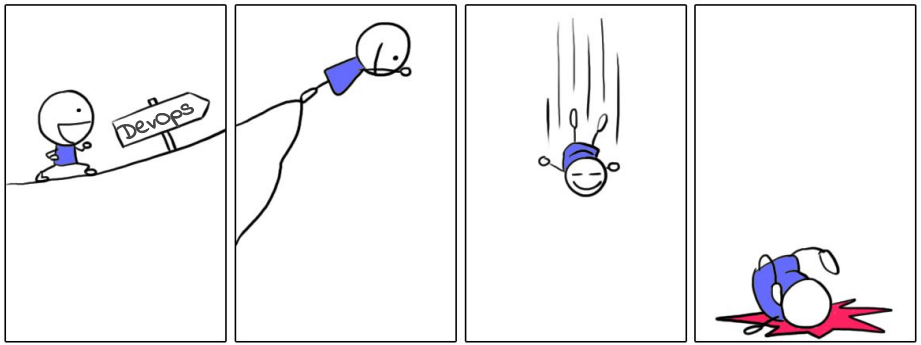
An intro to the magic of containers orchestration!

Catherin Cruz
@UserTwoGG

ScaleConfCo 2018
@scaleconfco



Why are we here?...



Another microservice, why are you like this?...

We need to rollback production, it's broken!...

Container orchestration, is it a DevOps thing?

Worked fine in my branch, DevOps problem now...

Ain't nobody got time for integration tests!

How Google does it?... it's **Black magic**, I knew it!

@UserTwoGG ScaleConfCo 2018

Monoliths vs Microservices

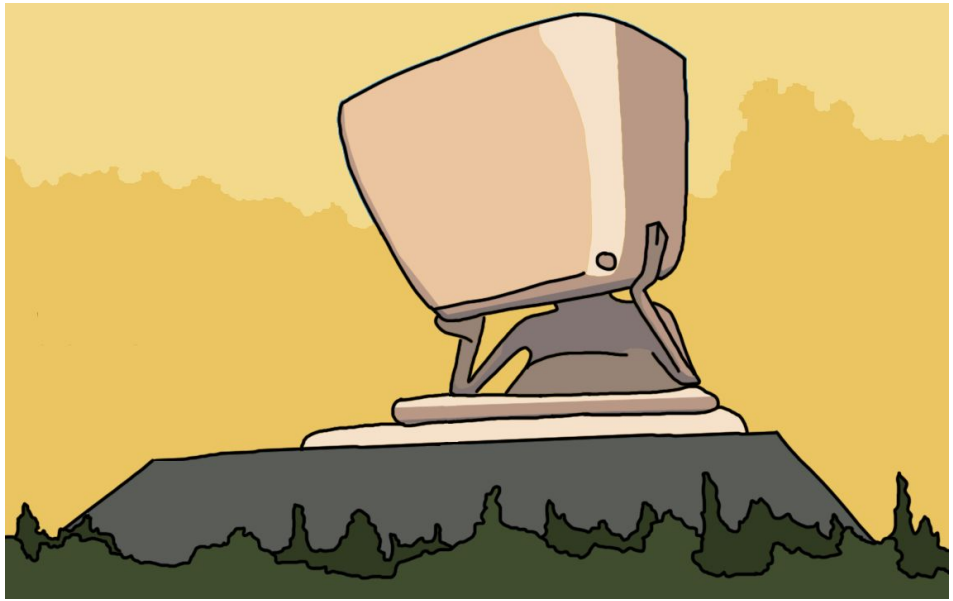
Monoliths had great feature sets, but...

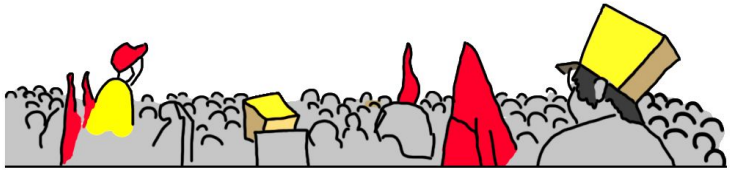
- Too many interdependent parts

- Integration and deployment issues

- Getting conflicting library versions

- Heavy network dependencies...





By divide them into **microservices** we can debug, update, and deploy without breaking the whole project.

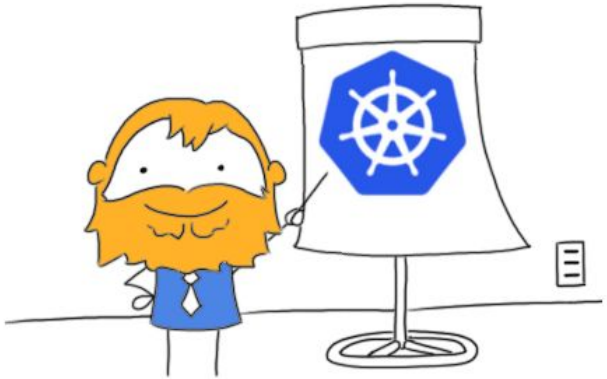
Containers

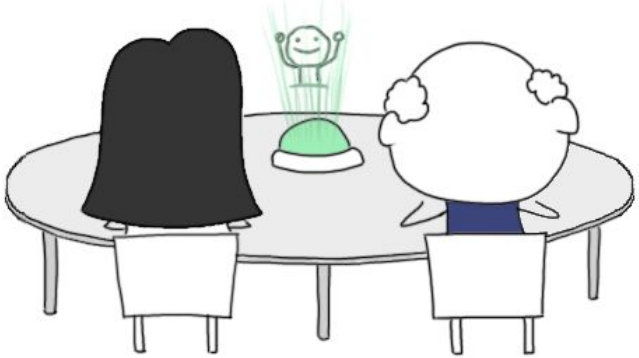
- Allows to specify exactly which parts of your environment you need.
- Easy to package, move, and often utilize compute resources more efficiently.
- Decouple an application and its dependencies from OS.

Docker

- Is primarily developed for Linux.
- Performs OS-level virtualization also known as containerization.
- It uses the resource isolation features of the Linux kernel.

What is Kubernetes



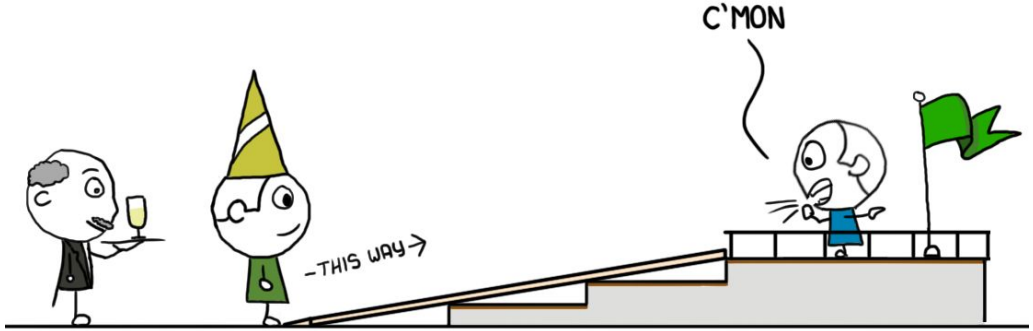


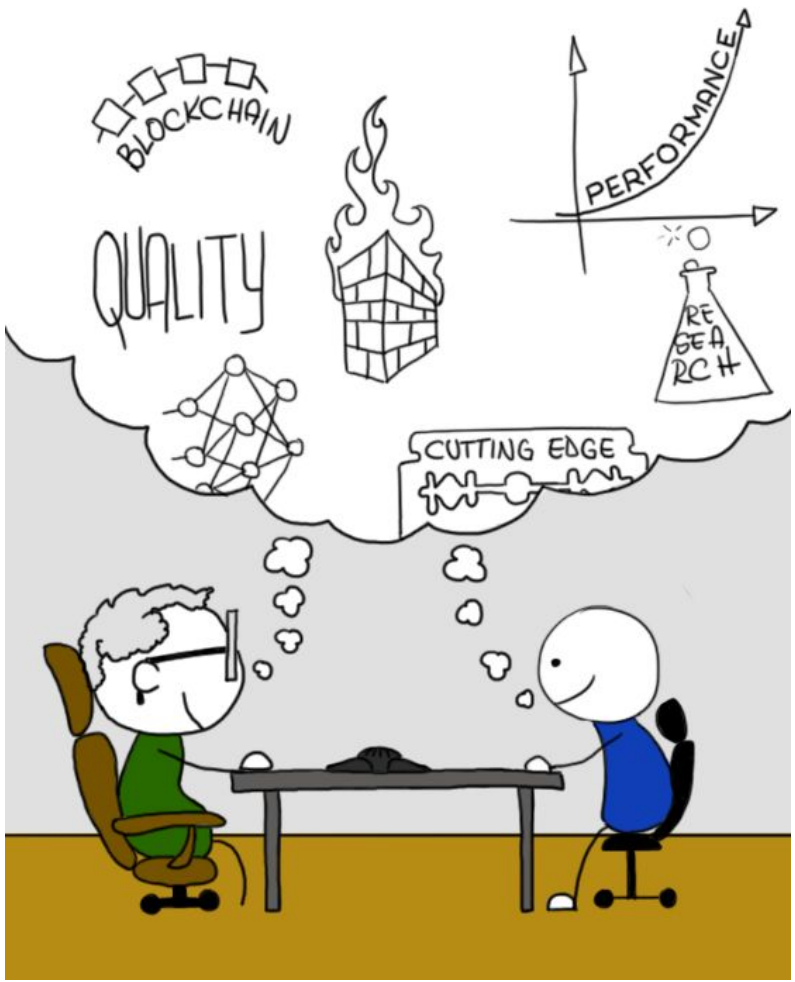
Is an open source orchestration system for **Docker containers**.

Manage containerized applications in a clustered environment.

Simplifies DevOps tasks such as debugging, deployment, scaling, configuration, versioning, and rolling updates.

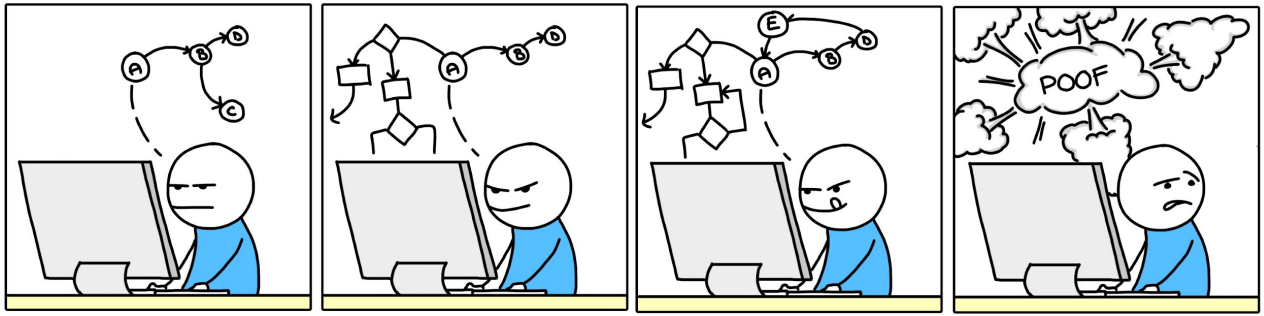
Uses labels as name tags to identify its objects.





Benefits

- ★ Container grouping using pod
- ★ Self-healing
- ★ Auto-scalability
- ★ DNS management
- ★ Load balancing
- ★ Rolling update or rollback
- ★ Resource monitoring and logging



Master Node

The controlling services in a Kubernetes cluster are basically the **components in charge**.

They monitor the cluster, make changes, schedule work, and respond to events, all this running on a single node.

Master Node: Kube-apiserver

It is the **brain to the master** and is front-end to the master or control plane.

Implements the RESTful API and consumes json via a manifest file.

Exposes an endpoint (port 443) so that kubectl (command line utility) can issue commands/queries and run on the master.

Master Node: Cluster Store

- ❏ Stateful
- ❏ Distributed
- ❏ Watchable

Build with **etcd**, an open source distributed key-value store.

Kubernetes uses etcd as the **source of truth** for the cluster. It takes care of storing and replicating data used by Kubernetes across the entire cluster. It is written in **Go language**.

Master Node: Kube-controller-manager

It is the controller of controllers, watches the shared state of the cluster and makes changes attempting to move the current state towards **the desired state**.

Kubernetes has a controller for: Replication, Endpoints, Namespaces, and Service accounts.

When a change is seen, reads the new information and implements the procedure that fulfills the desired state.

Master Node: Kube-scheduler

This is the process that watches API-server for new pods and assigns workloads to specific nodes in the cluster.

Tracks the resource utilization on each host to make sure that workloads are not scheduled in excess of the available resources.

Kubernetes Node

Kubelet

- The main Kubernetes agent on the node.
- Registers the node within the cluster
- Watches API server for work assignment.

Kube-proxy

- It is like the network brain of the node.
- Ensures every pod gets its own unique IP.
- Load balances across all pods in a service.

Objects: Pod

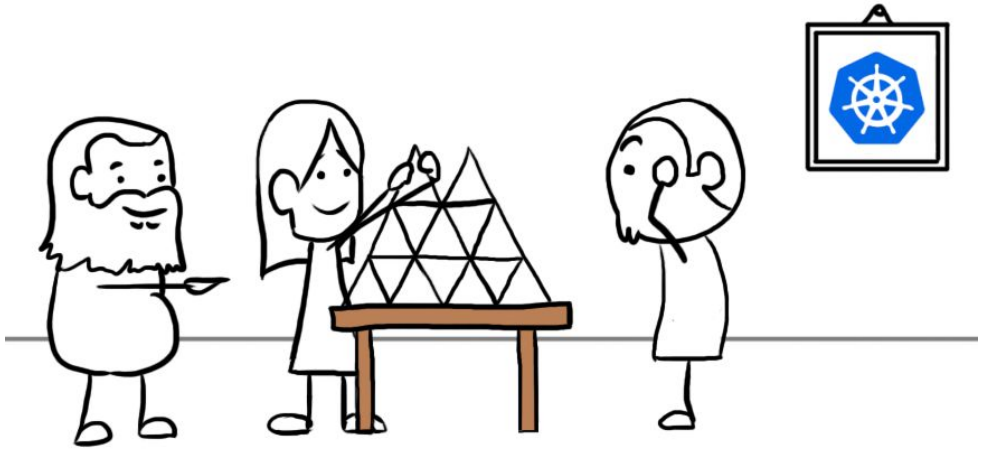
A pod is the basic building block of Kubernetes and is deployed as a single unit on a node in a cluster.

Every container in a Pod shares the network namespace, including the IP address and network ports.

A pod is a ring-fenced environment to run containers.

Each pod is assigned a unique IP address.

Pods are mortal and when they die they can not be resurrected.



Objects: Service

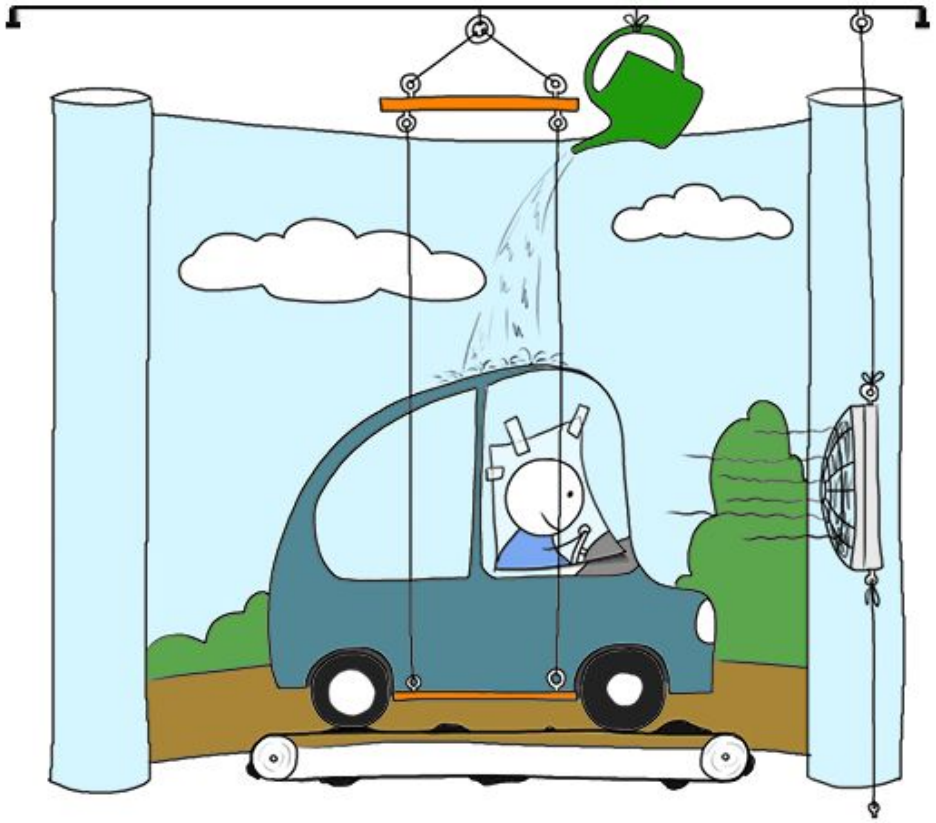
A service is like hiding multiple pods behind a network address.

Pods may come and go but the IP address and ports of your service remain the same.

Other applications can find your service through Kubernetes service discovery.

- ➔ Persistent
- ➔ Provide discovery
- ➔ Load balances
- ➔ Provide VIP layer

Objects: Volume



Represents a location where containers can store and access information.

When sharing files between containers, volume will outlive any container and all the data will be preserved across restarts.

For applications, volumes appear as part of a local file system.

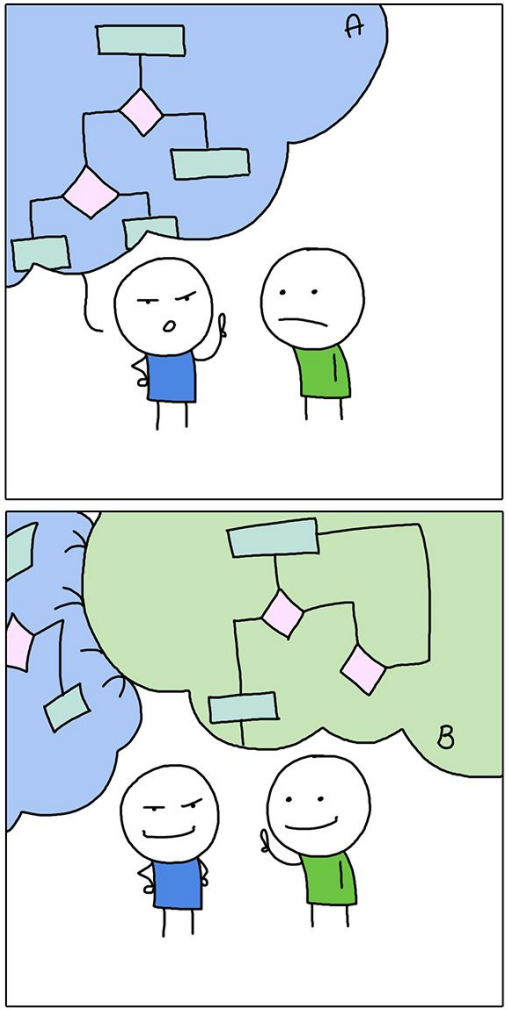
Objects: Namespaces

Namespace functions as grouping mechanism.

Services, pods, replication controllers, and volumes can easily cooperate within a namespace.

It provides a degree of isolation from other parts of the cluster, are intended for use in environments with many users spread across multiple teams, or projects.

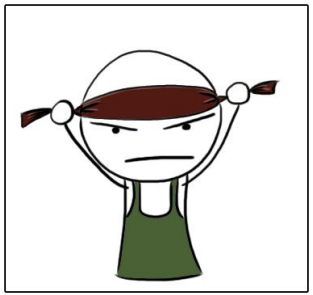
A way to divide cluster resources between multiple uses.

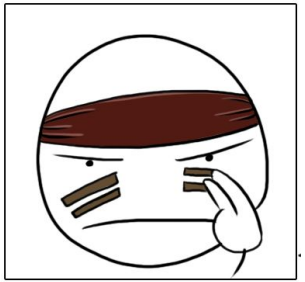


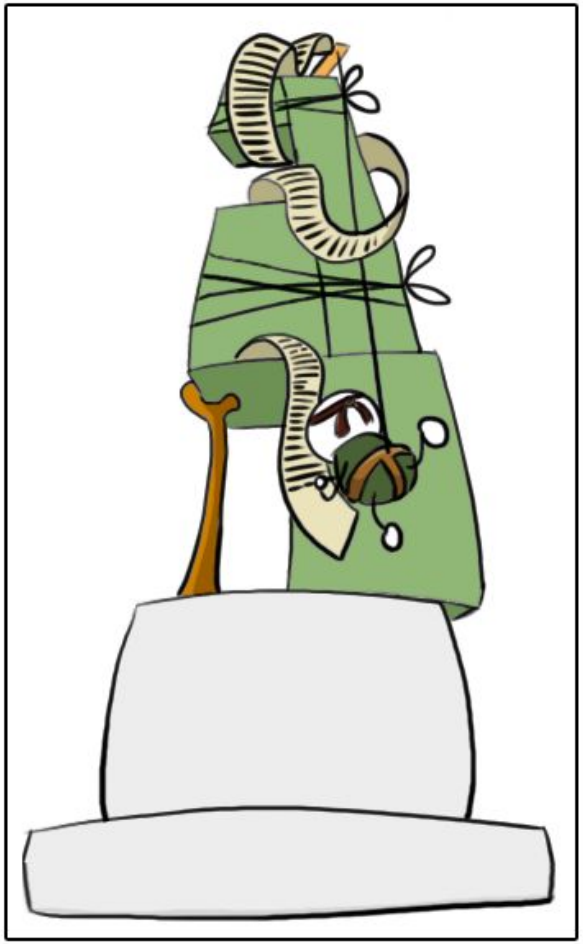
Kubectl commands

Now, we are going to use review the most used commands of Kubectl for running changes against Kubernetes clusters.

Also, If you want to get started with deploying containerized apps to Kubernetes, **Minikube** is the way to go, its helps you to deploy Kubernetes locally.



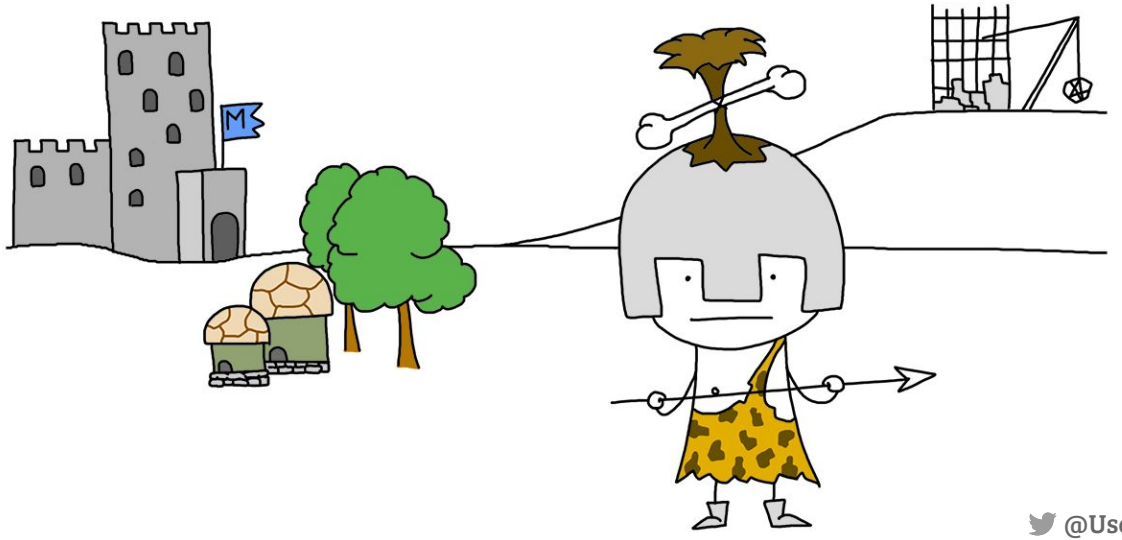


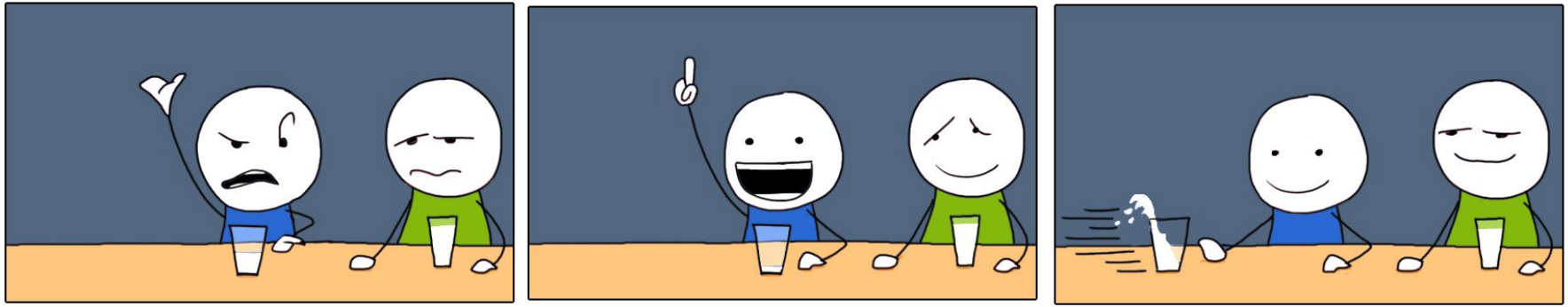


Create a deployment

```
$ kubectl run mydeploy --image=myhub/my_image:V1.4
```

```
$ kubectl expose deployment my_deploy --port=8080 --type=LoadBalancer
```

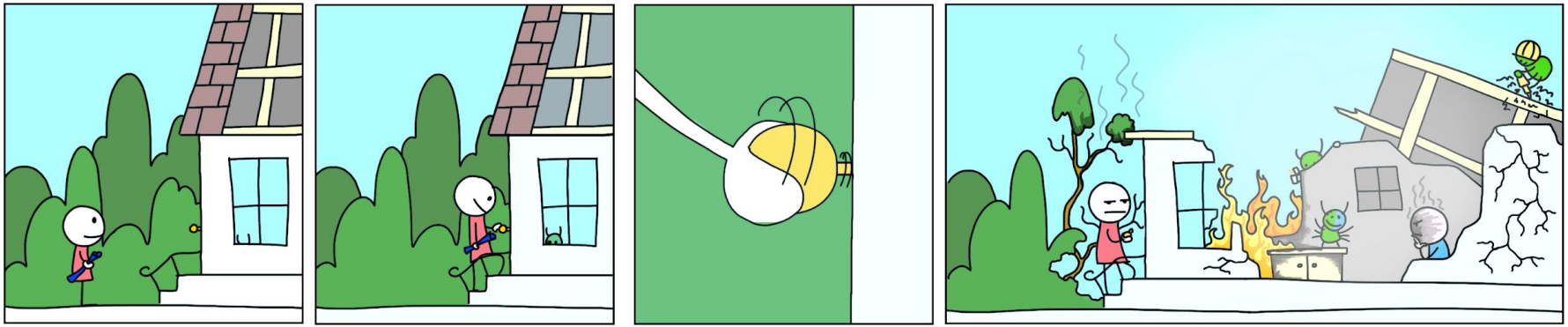




Scale a deployment

```
$ kubectl scale --replicas=3 deployment my_awesome_deployment
```

Kubernetes will help you to deploy the same way everywhere. You just need to say in a declarative language, how you'd like to run containers, a minimum amount of containers to running. Then you get the ability to scale up or down only when it's needed.



Rollback a deployment

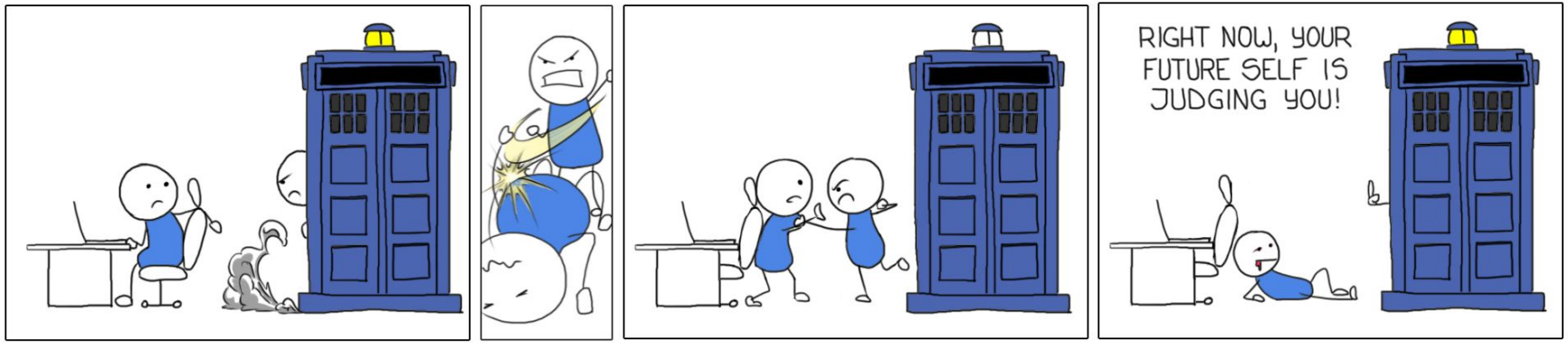
```
$ kubectl set image deployment mydeploy my_img=myhub/my_img:v1.5
```

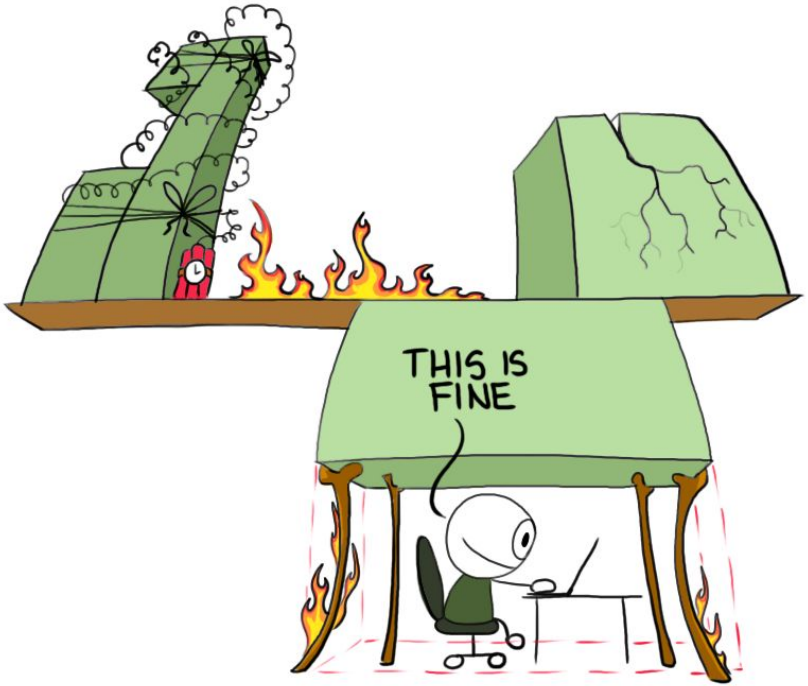
Sometimes you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping. By default, all of the Deployment's rollout history is kept in the system so that you can rollback anytime you want

Best practices!

- Use Helm Charts
- Keep Base Images Small
- Make sure your microservices aren't too micro
- Use plenty of descriptive labels
- Log everything to stdout and stderr
- One process per container

@UserTwoGG ScaleConfCo 2018





Link & References

Oficial Kubernetes Documentation
https://kubernetes.io/docs/home/

Kubernetes free playground
https://www.katacoda.com/courses/kubernetes/

Intro to Kubectl commands
https://github.com/twogg-git/k8s-intro

Images & media
http://www.monkeyuser.com/

Kubernetes

An intro to the magic of containers orchestration!

Catherin Cruz
@UserTwoGG

ScaleConfCo 2018
@scaleconfco