

# Lecture 13 – Kubernetes etc,

# Today

- Guest lecture
- A few words about the project
- Kubernetes

I NEED TO KNOW WHY MOVING OUR APP TO THE CLOUD DIDN'T AUTOMATICALLY SOLVE ALL OUR PROBLEMS.



Dilbert.com @ScottAdamsSays

YOU WOULDN'T LET ME RE-ARCHITECT THE APP TO BE CLOUD-NATIVE. JUST PUT IT IN CONTAINERS.



11-08-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel

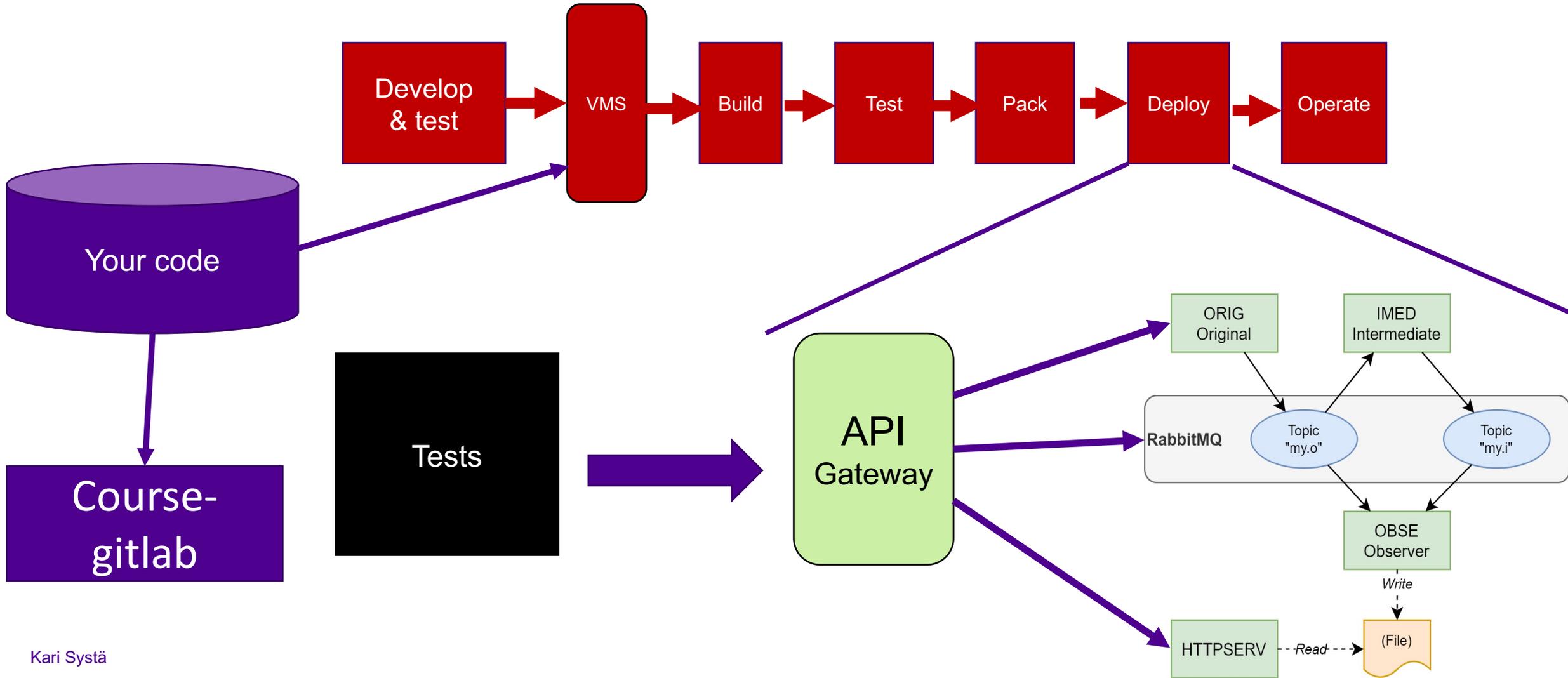
YOU CAN'T SOLVE A PROBLEM JUST BY SAYING TECHY THINGS. KUBERNETES.



# Project

## Gitlab

## Gitlab runner



# Starting both gitlab and runner together

- There are multiple ways and instructions on how to install gitlab-ci. Our recommendation is to use dockerized approach. The instructions typically assume that runner is deployed independently from gitlab. Internet has several tutorials on installing production grade set-up, but a few on how to install them together. For this project, starting both gitlab and runner makes sense. This discussion thread <https://gitlab.com/gitlab-org/gitlab-foss/issues/50851> discusses the issues and more importantly from there you can find two example files:
- Docker-compose.yml to start gitlab and runner
- Shell-script to register the runner with gitlab

version: '3.7'

services:

gitlab-web:

image: gitlab/gitlab-ce:latest  
container\_name: gitlab-web  
hostname: gitlab-web  
volumes:  
- './gitlab-config:/etc/gitlab'  
- './gitlab-logs:/var/log/gitlab'  
- './gitlab-data:/var/opt/gitlab'

ports:

- '2222:22'
- '8080:80'
- '443:443'
- '4567:4567'

environment:

```
GITLAB_OMNIBUS_CONFIG: |  
  gitlab_rails['gitlab_shell_ssh_port'] = 2222  
  registry_external_url 'http://localhost:4567'  
  registry['enable'] = true  
  unicorn['socket'] = '/opt/gitlab/var/unicorn/gitlab.socket'
```

networks:

- gitlab-network

gitlab-runner1:

image: gitlab/gitlab-runner:latest  
container\_name: gitlab-runner1  
hostname: gitlab-runner1  
volumes:  
- './gitlab-runner1-config:/etc/gitlab-runner:Z'  
- '/var/run/docker.sock:/var/run/docker.sock'

networks:

- gitlab-network

```
#!/bin/sh  
# Get the registration token from:  
# http://localhost:8080/root/${project}/settings/ci_cd
```

registration\_token=XXXXXXXXXXXXXXXXXXXX

```
docker exec -it gitlab-runner1 \  
  gitlab-runner register \  
    --non-interactive \  
    --registration-token ${registration_token} \  
    --locked=false \  
    --description docker-stable \  
    --url http://gitlab-web \  
    --executor docker \  
    --docker-image docker:stable \  
    --docker-volumes "/var/run/docker.sock:/var/run/docker.sock" \  
    --docker-network-mode gitlab-network
```

You may want to create your own version

To keep the required effort of assessment bearable, we assume

The system (Gitlab + runner + the extended application) can be started up with command sequence.

```
$ git clone -b project <the git url you gave>
```

```
$ cd <created folder>
```

```
$ docker-compose build --no-cache
```

```
$ docker-compose up -d
```

Testing running can be done with simple curl-commands and the API spec must be followed. In case you are unsure about the spec, please ask. For example,

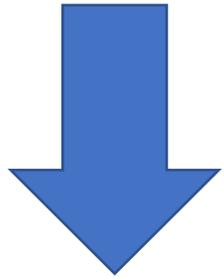
```
curl localhost:8083 -X put -d "PAUSED"
```

needs to work.

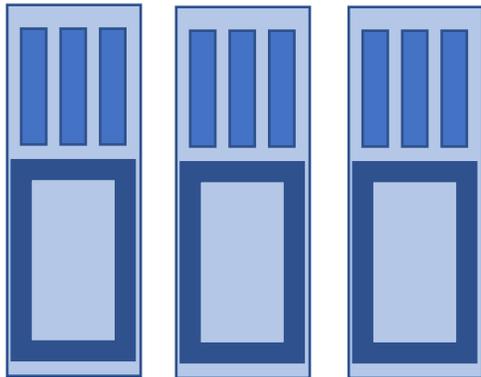
(It would be nice to test the gitlab CI. The assistant might try something, but we understand that the setup does not work for a cloned git repo out of the box.)

# Docker swarm - docker compose

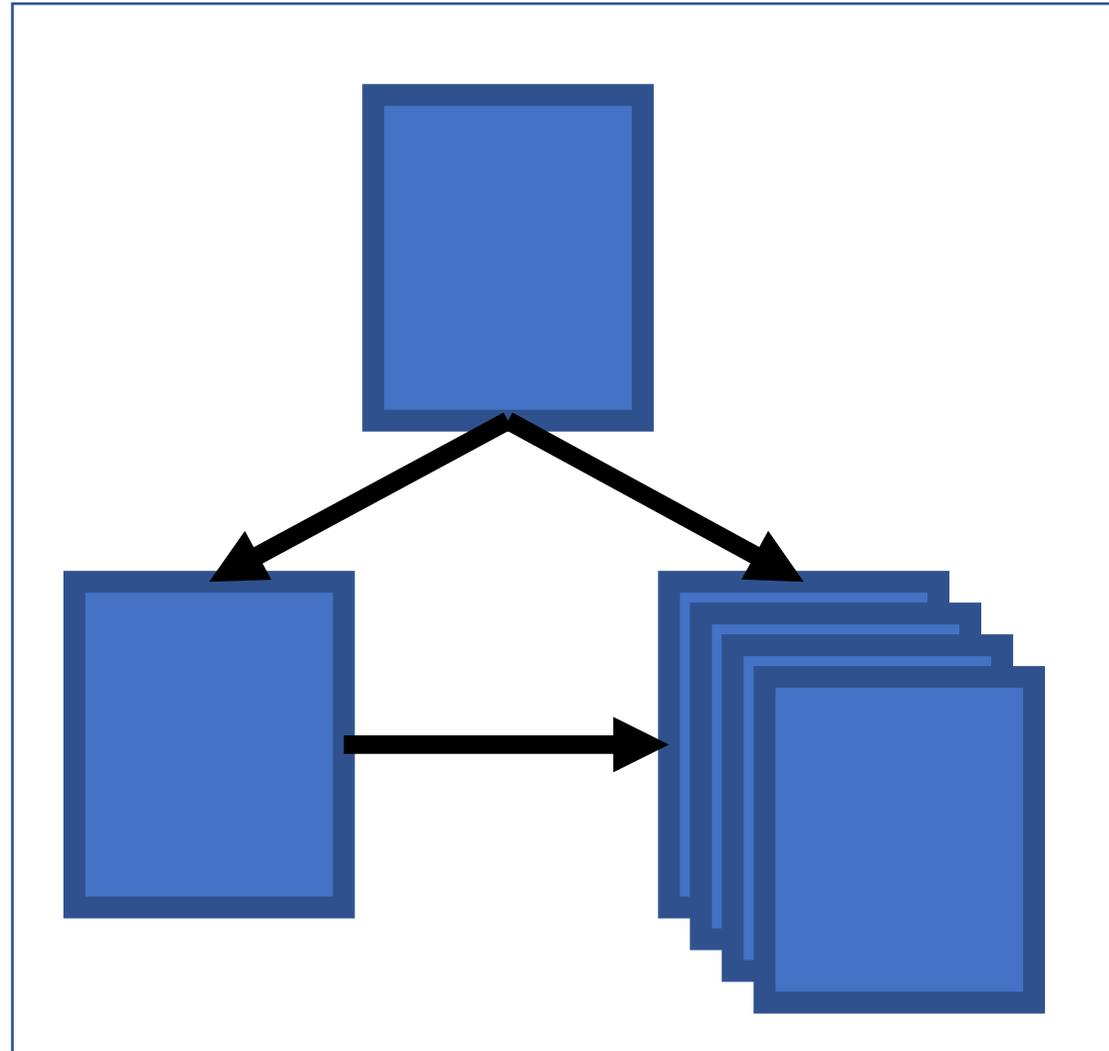
Orchestration



Docker swarm



Docker compose



# Kubernetes (k8s)

- An advanced tool for orchestration of containers (or other similar things)
- Comes with several features supporting automation and monitoring
- Has as steep learning curve
- Open source released by Google in 2004 (apache license)
- Now maintained by Cloud Native Computing Foundation (<https://www.cncf.io>)
- "platform for automating deployment, scaling, and operations of application containers across clusters of hosts"
- Is "job interview stuff"

# Modules of kubernetes.io tutorial

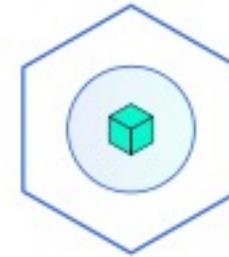
(source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/>)



1. Create a Kubernetes cluster



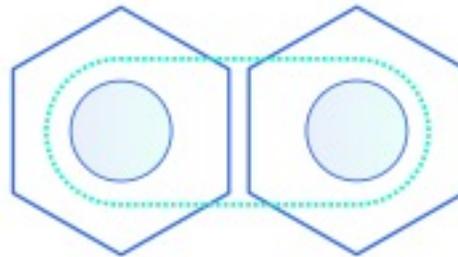
2. Deploy an app



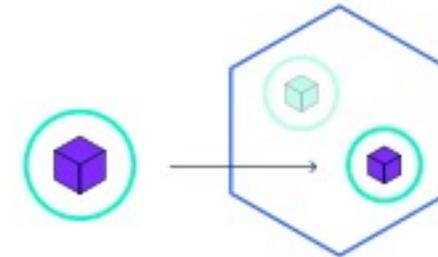
3. Explore your app



4. Expose your app publicly

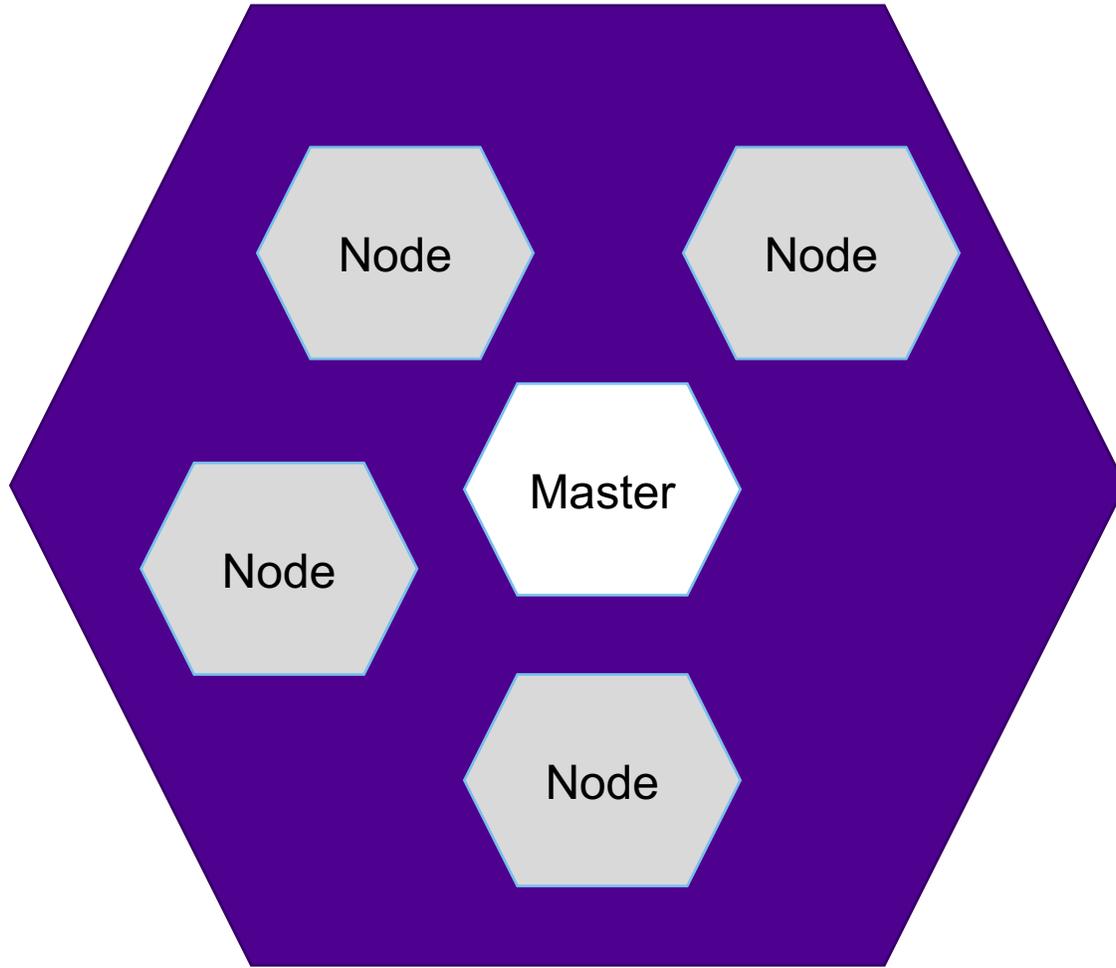


5. Scale up your app

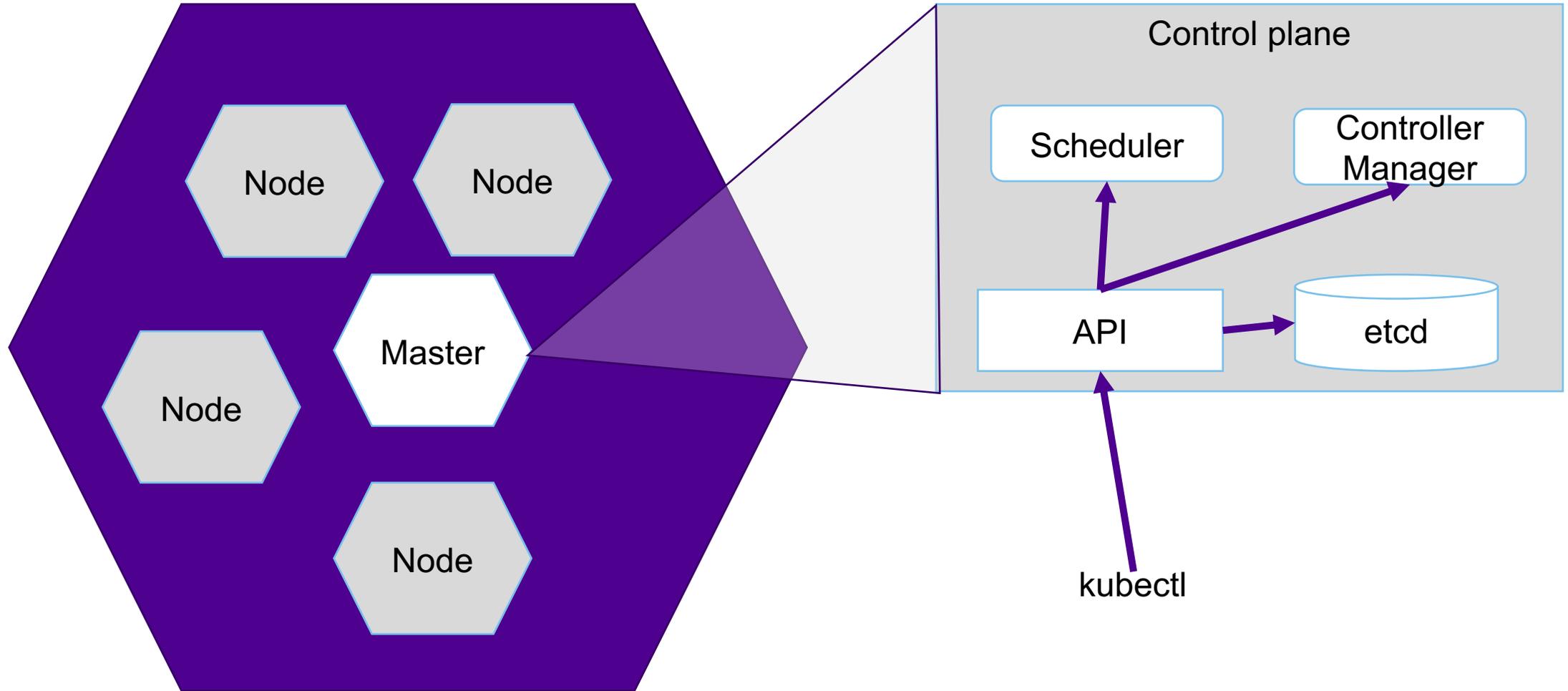


6. Update your app

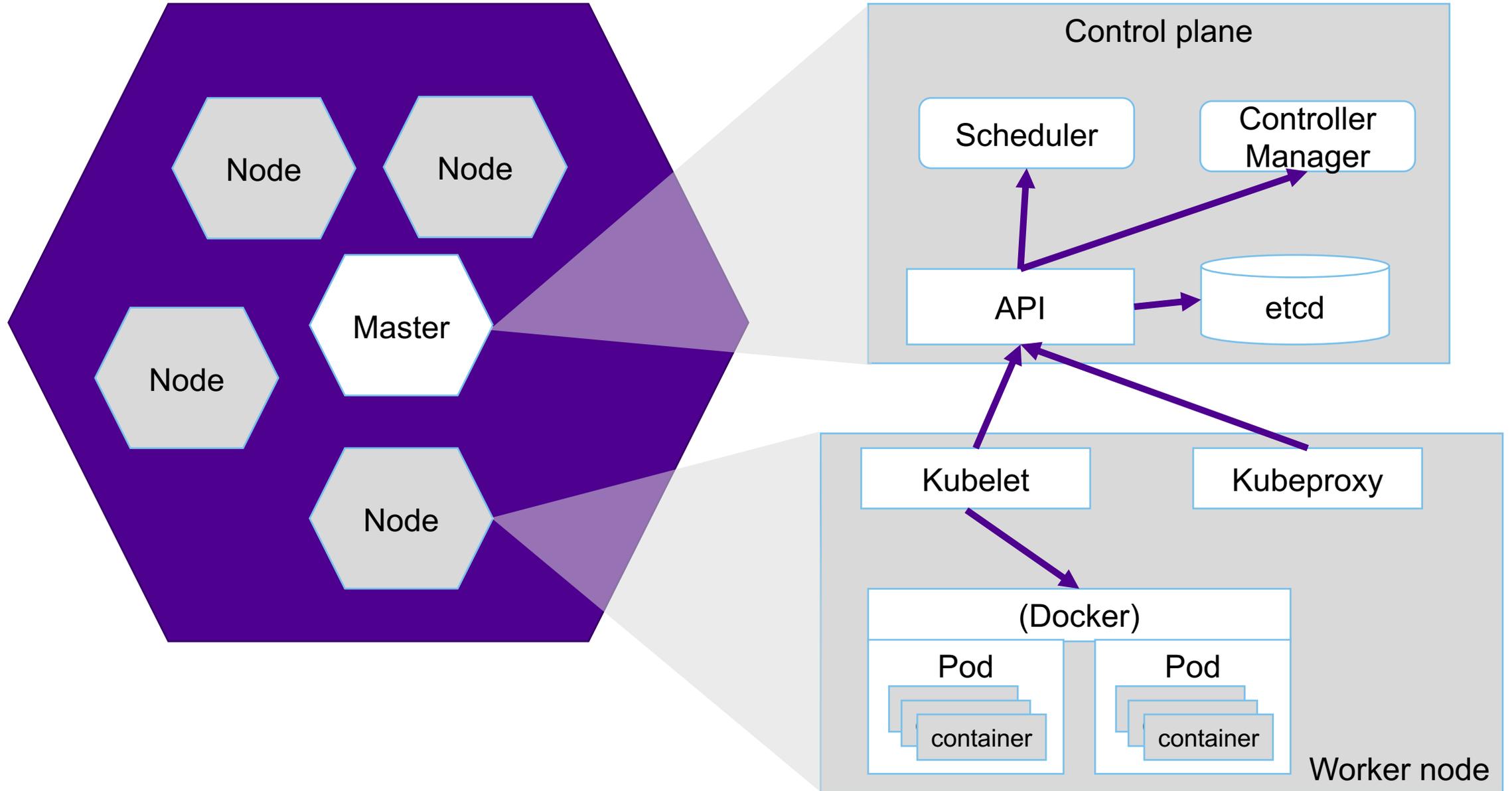
# Kubernetes Kluster



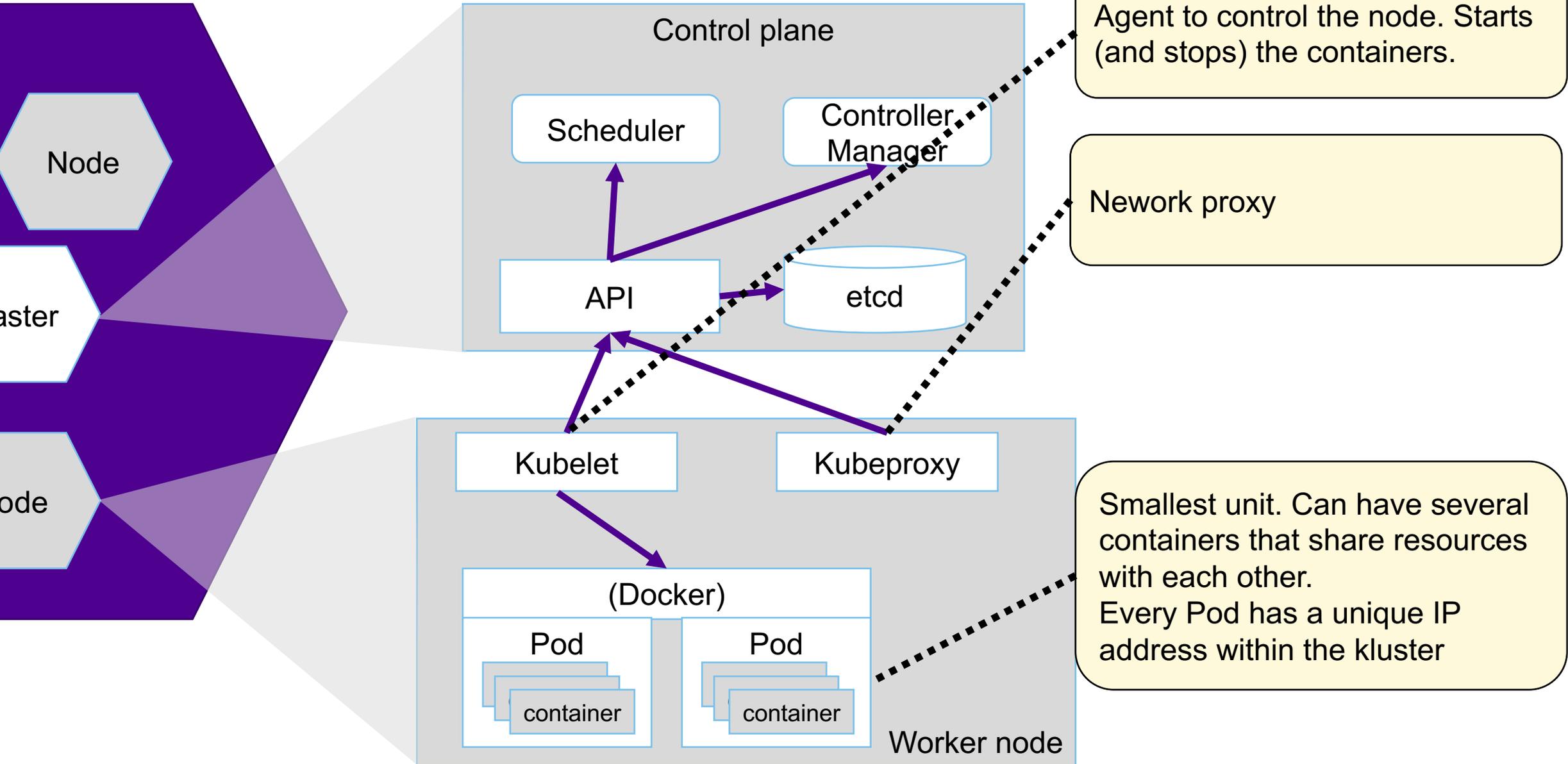
# Kubernetes Kluster



# Kubernetes Kluster



# Worker node

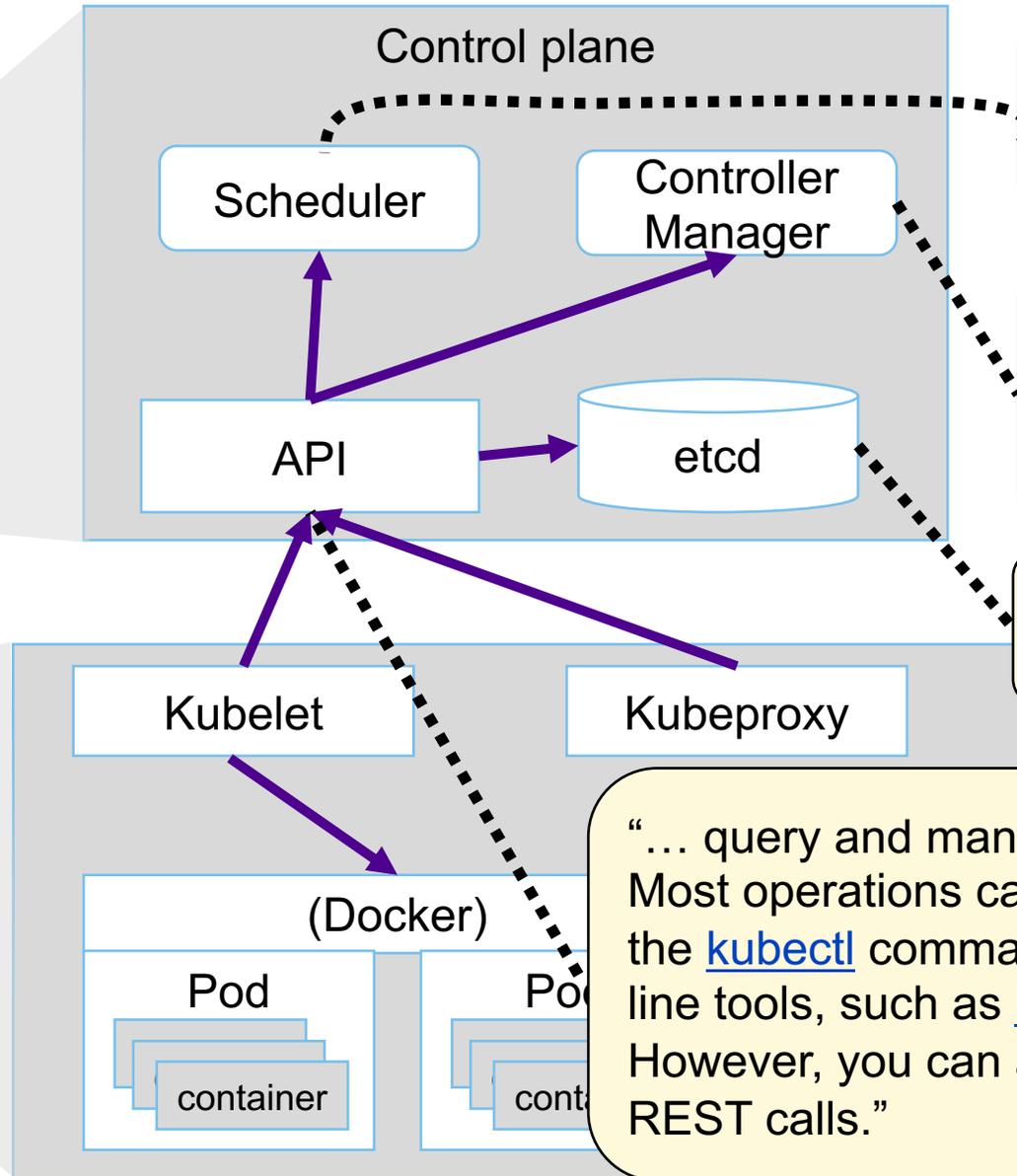
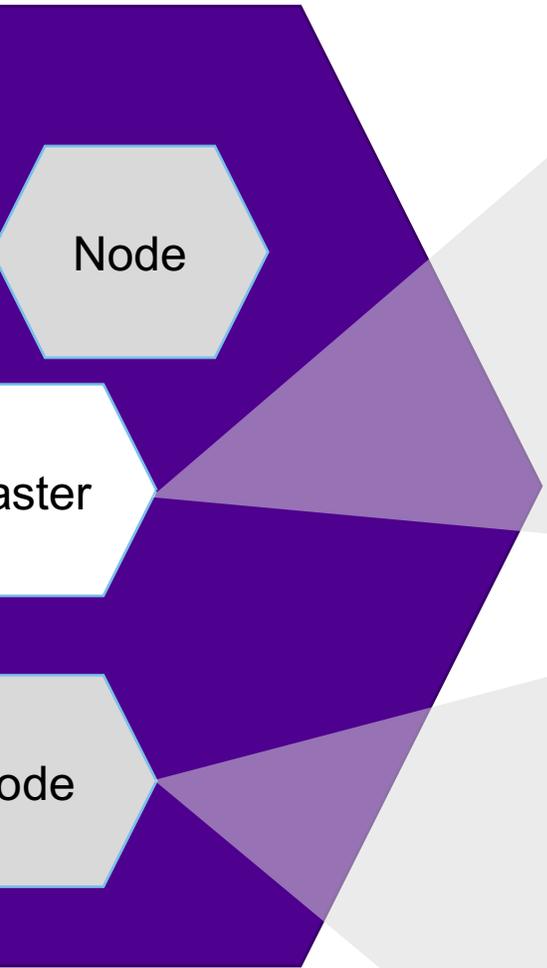


Agent to control the node. Starts (and stops) the containers.

Network proxy

Smallest unit. Can have several containers that share resources with each other. Every Pod has a unique IP address within the cluster

# Control plane



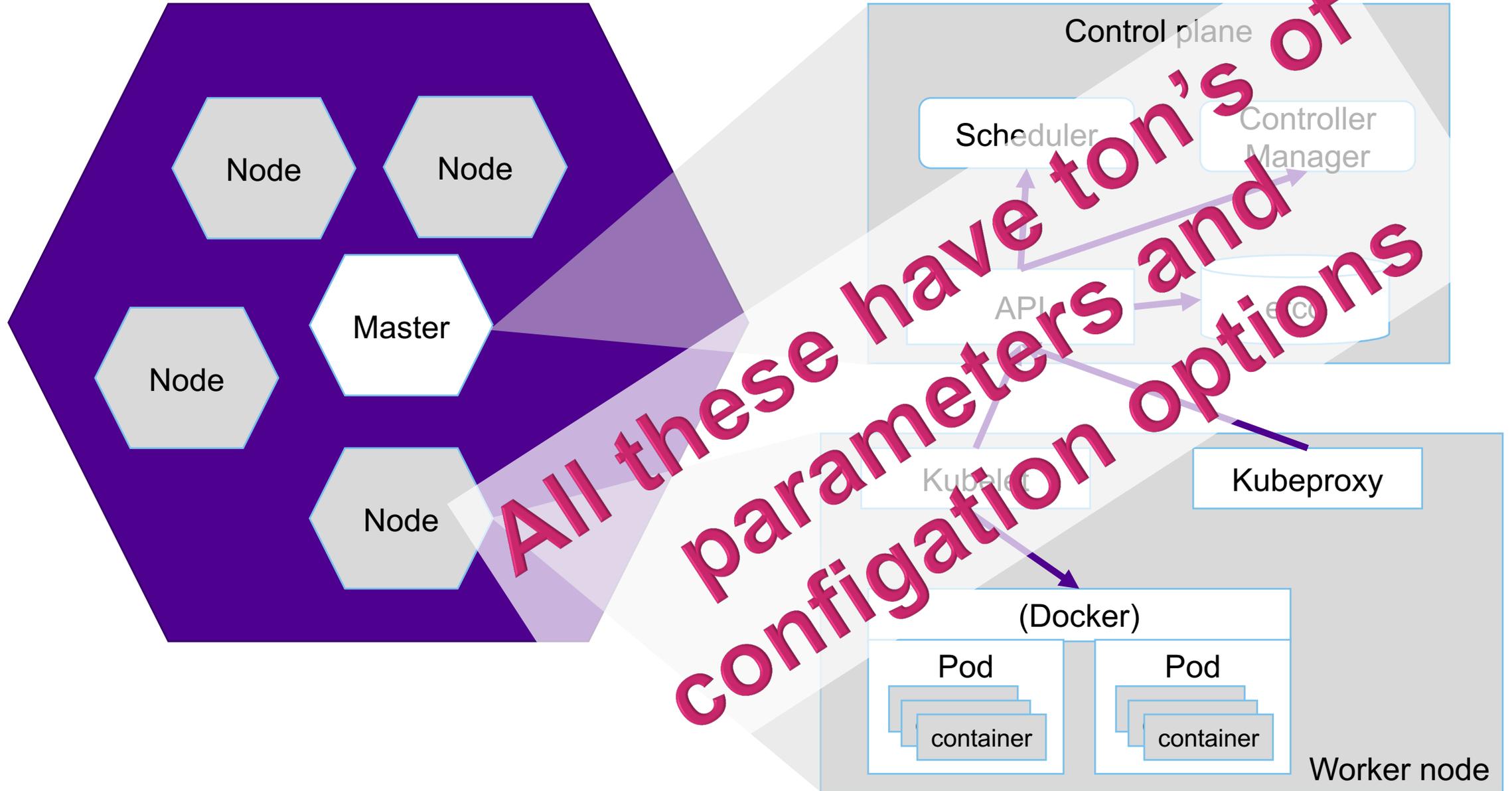
Sees that every Pod has a Node to run it.  
Finds the most suitable node

“ a control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state”

Key-value store for all kind of configuration information

“... query and manipulate the state of objects ... Most operations can be performed through the [kubectl](#) command-line interface or other command-line tools, such as [kubeadm](#), which in turn use the API. However, you can also access the API directly using REST calls.”

# Kubernetes Kluster



# Then next step

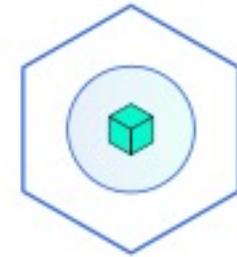
(source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/>)



1. Create a Kubernetes cluster



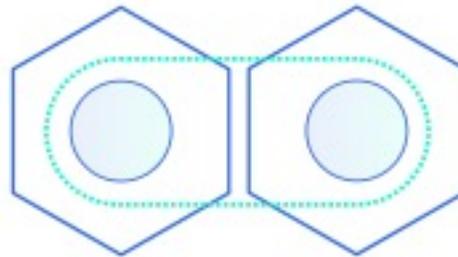
2. Deploy an app



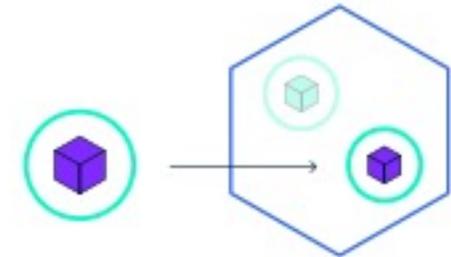
3. Explore your app



4. Expose your app publicly



5. Scale up your app



6. Update your app

# Deployment to Kubernetes

- Step 1: create Deployment (configuration)
- “A Deployment is responsible for creating and updating instances of your application”

```
$ kubectl apply -f
  https://k8s.io/examples/controllers/nginx-deployment.yaml
```

```
$ kubectl get deployments
```

| NAME             | READY | UP-TO-DATE | AVAILABLE | AGE |
|------------------|-------|------------|-----------|-----|
| nginx-deployment | 0/3   | 0          | 0         | 1s  |

```
$ kubectl get deployments
```

| NAME             | READY | UP-TO-DATE | AVAILABLE | AGE |
|------------------|-------|------------|-----------|-----|
| nginx-deployment | 3/3   | 3          | 3         | 18s |

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

# Then next step

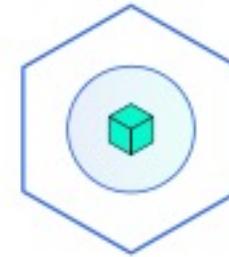
(source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/>)



1. Create a Kubernetes cluster



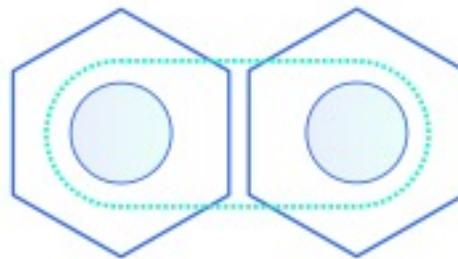
2. Deploy an app



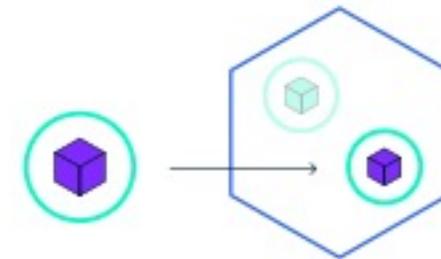
3. Explore your app



4. Expose your app publicly



5. Scale up your app



6. Update your app

# Examples of tools for trouble shooting

- **kubectl get** - list resources
- **kubectl describe** - show detailed information about a resource
- **kubectl logs** - print the logs from a container in a pod
- **kubectl exec** - execute a command on a container in a pod

# Then next step

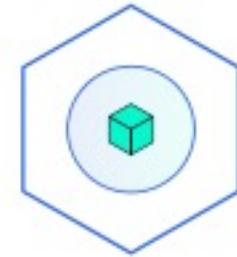
(source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/>)



1. Create a Kubernetes cluster



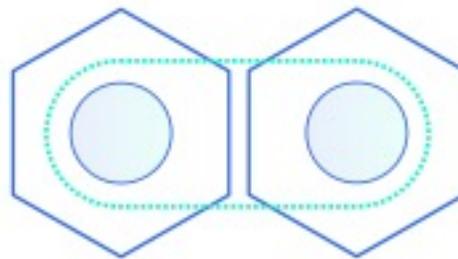
2. Deploy an app



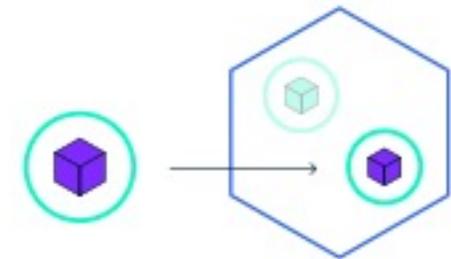
3. Explore your app



4. Expose your app publicly



5. Scale up your app



6. Update your app

# Kubernetes services

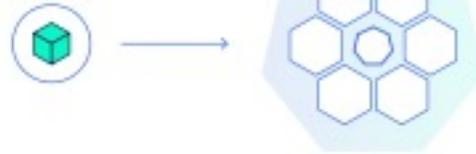
- Service
  - External IP and port
  - Load balancer
  - External name

# Then next step

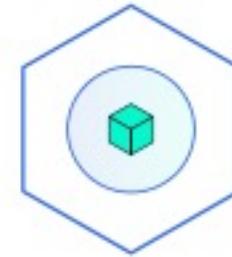
(source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/>)



1. Create a Kubernetes cluster



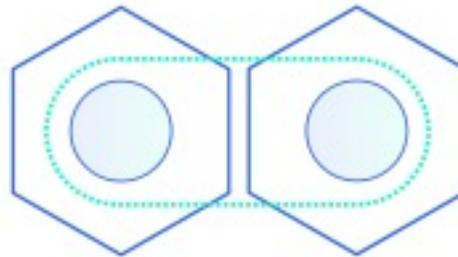
2. Deploy an app



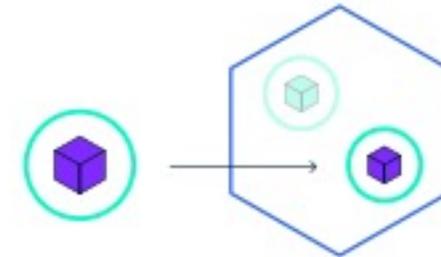
3. Explore your app



4. Expose your app publicly



5. Scale up your app



6. Update your app

# Scaling in Kubernetes

- Scaling out a Deployment increases the number of Pods to the new desired state.
- Kubernetes also supports auto scaling.
- Scaling to zero is also possible, and it will terminate all Pods of the specified Deployment.
- Services have an integrated load-balancer that will distribute network traffic to all Pods of an exposed Deployment.
- Services will monitor continuously the running Pods using endpoints, to ensure the traffic is sent only to available Pods.

# Then next step

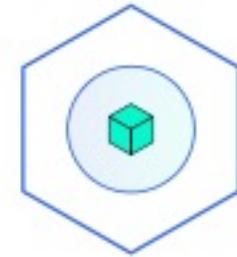
(source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/>)



1. Create a Kubernetes cluster



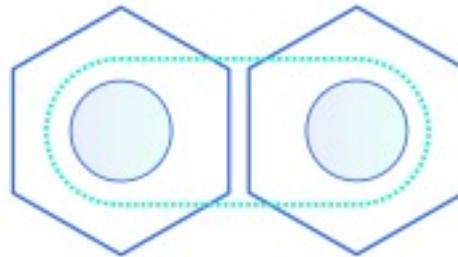
2. Deploy an app



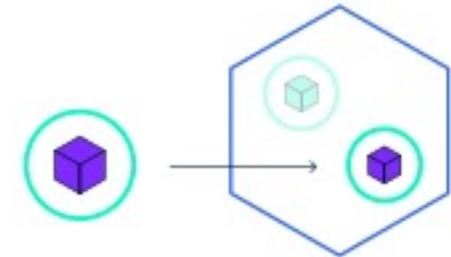
3. Explore your app



4. Expose your app publicly



5. Scale up your app



6. Update your app

# Updating an application

- Kubernetes uses **rolling updates**
- Zero down-time is the target

# According to the marketing material the supported features of Kubernetes include

- *Automatic binpacking.* Kubernetes automatically schedules the containers based on resource usage and constraints, without sacrificing the availability.
- *Self-healing.* Kubernetes automatically replaces and reschedules the containers from failed nodes. It also kills and restarts the containers which do not respond to health checks, based on existing rules/policy.
- *Horizontal scaling.* Kubernetes can automatically scale applications based on resource usage like CPU and memory. In some cases, it also supports dynamic scaling based on customer metrics.
- *Service discovery and Load balancing.* Kubernetes groups sets of containers and refers to them via a Domain Name System (DNS). This DNS is also called a Kubernetes service. Kubernetes can discover these services automatically, and load-balance requests between containers of a given service.
- *Automated rollouts and rollbacks.* Kubernetes can roll out and roll back new versions/configurations of an application, without introducing any downtime.
- *Secrets and configuration management.* Kubernetes can manage secrets and configuration details for an application without re-building the respective images. With secrets, we can share confidential information to our application without exposing it to the stack configuration, like on GitHub.
- *Storage orchestration.* With Kubernetes and its plugins, we can automatically mount local, external, and storage solutions to the containers in a seamless manner, based on software-defined storage (SDS).
- *Batch execution.* Besides long running jobs, Kubernetes also supports batch execution.

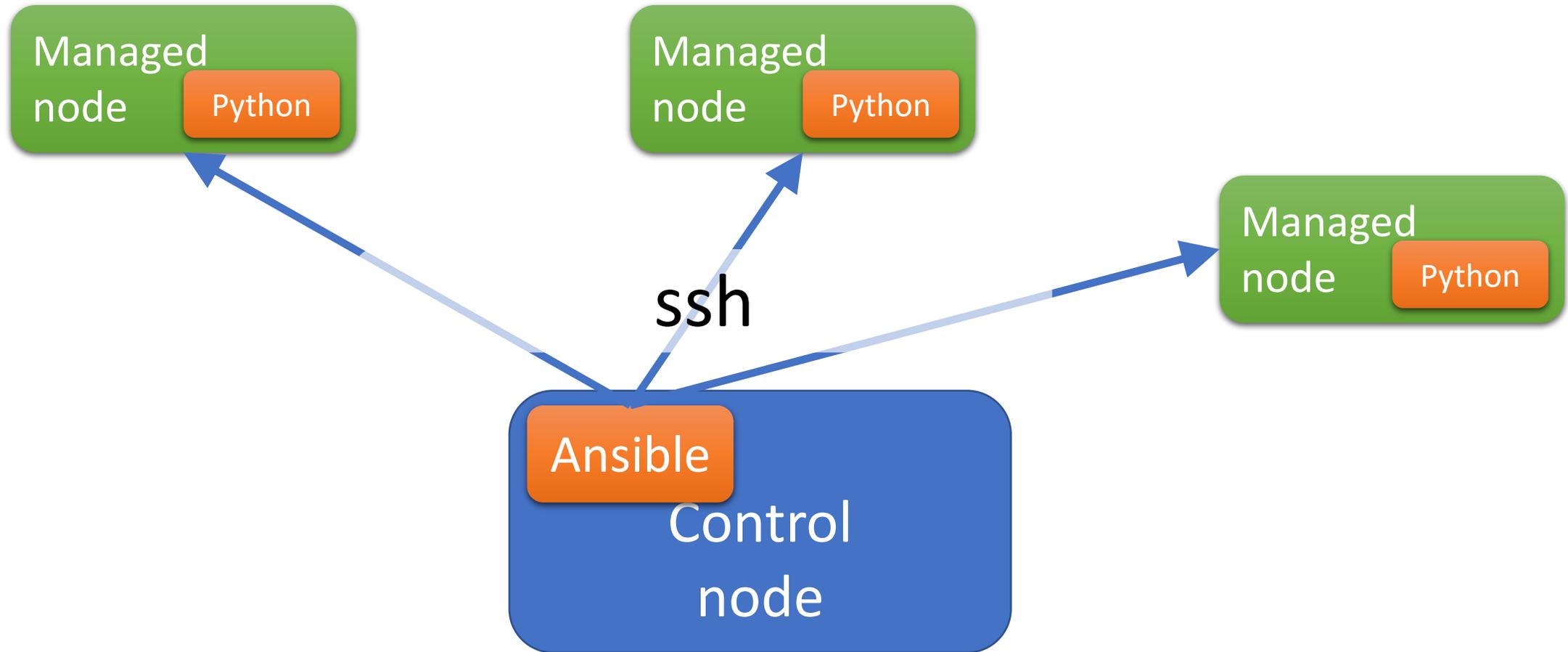
# BSc thesis of Petteri Strömberg

**Table 1.** Functional differences between Docker Swarm and Kubernetes.

| Layer               | Feature            | Docker Swarm | Kubernetes |
|---------------------|--------------------|--------------|------------|
| Resource Management | Memory             | +            | +          |
|                     | CPU                | +            | +          |
|                     | Local Volumes      | +            | +          |
|                     | Remote Volumes     | +            | ++         |
|                     | Networking         | +            | +          |
| Scheduling          | Placement          | +            | +          |
|                     | Scaling            | +            | +          |
|                     | Automatic Scaling  |              | +          |
|                     | Readiness Checking |              | +          |
|                     | Restarting         | +            | +          |
|                     | Rolling Updates    | +            | +          |
|                     | Co-location        |              | +          |
| Service Management  | Labels             | +            | +          |
|                     | Groups             |              | +          |
|                     | Load Balancing     | +            | ++         |

+ basic functionality, ++ advanced functionality

# Reminder about Ansible



# Other alternatives to Ansible

- **Chef:** Chef is written in the Ruby programming language and its CLI uses a Ruby-based DSL. Chef assumes an agent in the controlled nodes.
- **SaltStack:** Written in Python, SaltStack(Salt) holds your inventory's state on a master server, with YAML being the default format for storing configurations. SaltStack templates use the Jinja templating language, which will be familiar to Python users.
- **Puppet:** Another tool on the configuration management side, Puppet requires a master server, called the Puppet master, which stores the configuration of your infrastructure and pushes changes out to clients.
- **Terraform:** A different approach to both Ansible and Chef. Terraform focuses on setting up your entire infrastructure and provisioning servers. As such, it falls on the orchestration side of the spectrum, but it can be used alongside configuration focused tools like Chef and Ansible

# Example of an alternative approach: CloudFoundry

- **Claim:** “*Cloud Foundry makes going from code to running apps as easy as a single cf push command. Don’t spend your time writing infrastructure config for Kubernetes and Istio. Stay focused on your code.*”
- “First, equating the Cloud Foundry experience to a Kubernetes experience is like equating apples and walnuts. They’re not at all related. Kubernetes is all about being an infrastructure abstraction, but that’s not optimized for developers, and it has a more broadly applicable set of use cases – you can take a legacy app, slap it into a container and run it on Kubernetes; you could craft your own containers that are for a more modern architecture and run that on Kubernetes; and a whole bunch of things in between. The Cloud Foundry experience is focused on optimizing for the developer that’s writing custom software for business or, in many cases, government applications. It’s all about custom code.”
- CloudFoundry may run on top of Kubernetes

# About the future

- I suspect that the programmatic and simpler approaches are coming.
- CDK presented in the guest lecture last week is an example of that development.

# Next week

- Recap and exam material

# Terraform

- Declarative approach, too.
  - User writes a specification of the target state
- Pluggable architecture
  - Can have providers for AWS, Azure, gCloud
- Code – (plan – )apply

example.tf

```
terraform {  
  required_providers {  
    aws = {  
      source = "hashicorp/aws"  
      version = "~> 2.70"  
    }  
  }  
}
```

```
provider "aws" {  
  profile = "default"  
  region = "us-west-2"  
}
```

```
resource "aws_instance" "example" {  
  ami = "ami-830c94e3"  
  instance_type = "t2.micro"  
}
```

```
$ terraform show  
# aws_instance.example:  
resource "aws_instance" "example" {  
  ami = "ami-830c94e3"  
  arn = "arn:aws:ec2:us-east-1:130490850807:instance/i-0bbf06244e44211d1"  
  associate_public_ip_address = true  
  availability_zone = "us-west-2"  
  cpu_core_count = 1  
  cpu_threads_per_core = 1  
  disable_api_termination = false  
  ebs_optimized = false  
  get_password_data = false  
  id = "i-0bbf06244e44211d1"  
  instance_state = "running"  
  instance_type = "t2.micro"  
  ipv6_address_count = 0  
  ipv6_addresses = []  
  monitoring = false  
  primary_network_interface_id = "eni-0f1ce5bdae258b015"  
  private_dns = "ip-172-31-61-141.ec2.internal"  
  private_ip = "172.31.61.141"  
  public_dns = "ec2-54-166-19-naws.com"  
  public_ip = "54.166.19.244"  
  security_groups = [  
    "default",  
  ]  
  source_dest_check = true  
  subnet_id = "subnet-1facdf35"  
  tenancy = "default"  
  volume_tags = {}  
  vpc_security_group_ids = [  
    "sg-5255f429",  
  ]  
}
```

# From Wikipedia

| Tool                           | Released by            | Method        | Approach                   | Written in                   | Comments         |
|--------------------------------|------------------------|---------------|----------------------------|------------------------------|------------------|
| <b>Pulumi</b>                  | Pulumi <sup>[13]</sup> | Push          | Declarative                | Typescript, Python, Go       |                  |
| <b>Chef</b>                    | Chef (2009)            | Pull          | Declarative and imperative | Ruby                         |                  |
| <b>Otter</b>                   | Inedo                  | Push          | Declarative and imperative | -                            | Windows oriented |
| <b>Puppet</b>                  | Puppet (2005)          | Pull          | Declarative                | C++ & Clojure from 4.0, Ruby |                  |
| <b>SaltStack</b>               | SaltStack              | Push and Pull | Declarative and imperative | Python                       |                  |
| <b>CFEngine</b>                | CFEngine               | Pull          | Declarative                | -                            |                  |
| <b>Terraform</b>               | HashiCorp (2014)       | Push          | Declarative                | Go                           |                  |
| <b>DSC</b>                     | Microsoft              | Push/Pull     | Declarative/Imperative     | PowerShell                   |                  |
| <b>Ansible / Ansible Tower</b> | RedHat (2012)          | Push          | Declarative and imperative | Python                       |                  |

Other tools include [AWS CloudFormation](#), [cdist](#), [StackStorm](#) and [Juju \(software\)](#).

# Another: Heroku

<https://devcenter.heroku.com/articles/how-heroku-works>

- “Heroku lets you deploy, run and manage applications written in Ruby, Node.js, Java, Python, Clojure, Scala, Go and PHP.”
- “Heroku is a polyglot platform – it lets you build, run and scale applications in a similar manner across all the languages – utilizing the dependencies and Procfile. The Procfile exposes an architectural aspect of your application...”
- Architectural principles:
  - Strict separation of code and configuration, explicit dependency declaration, tight development iterations and parity between environments.
  - Applications are run as independent, lightweight, and stateless processes with quick startup and shutdown.
  - Execute auxiliary tasks in one-off processes and view application output via collated log-stream.

# Automation challenges

- "...provisioning scripts were considered error-prone and, according to developers, they did not work in some environments..."
- "...automation of the network in was said to be difficult in addition to dealing with legacy system..."
- "Networks are pretty hard. Some of the databases are pretty hard too because the old relational databases haven't been designed to be clustered..."

## Reminders about infrastructure as code

**All SW engineering principles should be applied.**

- Testing
- Maintenance
- Documentation
- Version and configuration management
  
- Bugs may stop the whole engine



## BLE OF DEVOPS TOOLS (V3)

EMBED [



|                            |                          |                                      |                                      |                           |                             |                          |
|----------------------------|--------------------------|--------------------------------------|--------------------------------------|---------------------------|-----------------------------|--------------------------|
| 23<br>Os<br>Fn<br>FitNesse | 24<br>Fr<br>Ju<br>JUnit  | 25<br>Fr<br>Ka<br>Karma              | 26<br>Os<br>Su<br>SoapUI             | 27<br>En<br>Ch<br>Chef    | 28<br>Fr<br>Tf<br>Terraform | 29<br>Fr<br>X<br>Xe      |
| 41<br>Fr<br>Se<br>Selenium | 42<br>Fr<br>Jm<br>JMeter | 43<br>Os<br>Ja<br>Jasmine            | 44<br>Pd<br>Sl<br>Sauce Labs         | 45<br>Os<br>An<br>Ansible | 46<br>Os<br>Ru<br>Rudder    | 47<br>Os<br>C<br>C       |
| 59<br>Os<br>Ga<br>Gatling  | 60<br>Fr<br>Tn<br>TestNG | 61<br>Fm<br>Tt<br>Tricentis<br>Tosca | 62<br>Pd<br>Pe<br>Perfecto           | 63<br>En<br>Pu<br>Puppet  | 64<br>Os<br>Pa<br>Packer    | 65<br>Os<br>C<br>AV<br>C |
| 77<br>Fr<br>Cu<br>Cucumber | 78<br>Os<br>Mc<br>Mocha  | 79<br>Os<br>Lo<br>Locust.io          | 80<br>En<br>Mf<br>Micro Focus<br>UFT | 81<br>Os<br>Sl<br>Salt    | 82<br>Os<br>Ce<br>CFEngine  | 83<br>Os<br>E<br>E       |

# From:

<https://tutorials.cloudfoundry.org/cf-and-k8s/docs/comparing/>

## Design Approach

- **Cloud Foundry** is an **opinionated set of components** that are designed and distributed to work together.
- **Kubernetes** is a **flexible and extensible** system with a wide range of open source components from which you can choose, install, configure and maintain.

## Built-in Functionality

- **Cloud Foundry** embodies an **opinionated workflow**, and automates the building of container images from application code, configuration of HTTPS access to your apps, and comes pre-configured with multi-tenancy access controls that are suitable for use in banks and governments.
- **Kubernetes** offers a considerable amount of flexibility, and **can be configured and extended to support virtually any workflow**. As such, you could assemble your own set of components on Kubernetes that replicate the functionality of Cloud Foundry.