Welcome to the

Bandung, Indonesia

2018 Global Azure BOOTCAMP

A BIG thank you to the 2018 Global Sponsors!

Easy Deployment with Azure Container Service (AKS)

```
$ whoami
```

Alwin Arrasyid

winter@dycode.com | alwin.ridd@gmail.com

twitter: @alwin_wint3r

github: alwint3r

Lead Software Engineer

x@dycode.com | http://dycodex.com

I'll talk about...

- Deployment prior to container era (based on my experience)
- Containerization Era
- Azure Container Service (AKS)

Deployment Prior to Container Era

based on my experiences

Deployment Prior to Container Era

- Applications are running direclty on top of virtual machines' OS
- Developers / sysadmin are required to install the dependencies by themselves
- Developers also required to have the knowledge of tools like systemd, sysVinit, pm2.

Deployment Prior to Container Era: The Problems

- Conflicting dependencies
  - Two different applications have the same dependencies with different versions, for example:
    - Application A requires Node.js <= v0.12.x
    - Application B requires Node.js >= v8.x.x
- Manual virtual machine setup can be painful
  - Installing required softwares with correct version
  - Managing deployment keys if you are using private git repository

Deployment Prior to Container Era

- Ansible Playbook is good for automating virtual machine setup and deployment.
  - Has *module* for Azure and tons of others modules.
- Developers / sysadmin are required to have the knowledge of ansible modules.

Deployment Prior to Container Era - The Problems

- Scaling is hard
  - The easiest way to achieve scaling is by doing vertical scaling
    - Upgrading resources is expensive.
  - Scaling out means setting up and managing more virtual machines
  - How do we load-balance?

"It works on my computer"

- Every developer, at least once in their life.

Containerization Era

Containerization Era

- Before Docker famous, container technologies such as OpenVZ and LXC has been around since mid 2000s
- Docker is released in 2013 and has been adopted widely in the industry ever since

Docker what?

- Docker enables true independence between applications and infrastructure
  - No more conflicting dependencies between applications
- Containerized applications environment are more consistent, regardless of what kind of OS hosting them
  - Bye "it works on my machine" excuse

Using Docker

- Containerizing existing application with Docker is easy as pie
- Using a file called *Dockerfile* tas the recipe for our Docker *image* which will be the base of our *containers*

```
FROM node:8-alpine
WORKDIR /usr/src/app
COPY . .
RUN npm install --silent
EXPOSE 8080
CMD ["node", "app.js"]
```

Using Docker

- Building image from Dockerfile is as easy as typing

```
docker build -t node-app:latest .
```

- Running container is also as easy as typing

```
docker run --name my-node-app --port=8080:8080 node-app:latest
```

```
docker run --name my-node-app-2 --port=8081:8080 node-app:latest
```

Using Docker

- To summarise, here's the basic Docker workflow:
  - Write Dockerfile
  - Build image from Dockerfile
  - Run container from that image
    - Bind container port to host OS port
    - Create *volume* in host OS filesystem and link it to our container filesystem to persist data

Using Docker Compose

- Building microservices? *docker-compose* got you covered
- Use docker-compose to group the inter-linked containers
  - Containers properties and connection between them are described by a *docker-compose.yml* file
  - Containers created by docker-compose are on the same network
    - Easy ICC (Inter-Container Communication)

docker-compose.yml example

```
version: "3"
services:
  rest_api:
    image: my-node-app:latest
    ports:
      - "8080:8080"
    environment:
      MONGODB_URI: mongodb://mongodb/rest_api_db

  mongodb:
       image: mongo:3.2-jessie
       ports:
         - "27017:27017"
```

Problems with Docker

- How do we scale?
- How do we load-balance containers?

Introducing Kubernetes

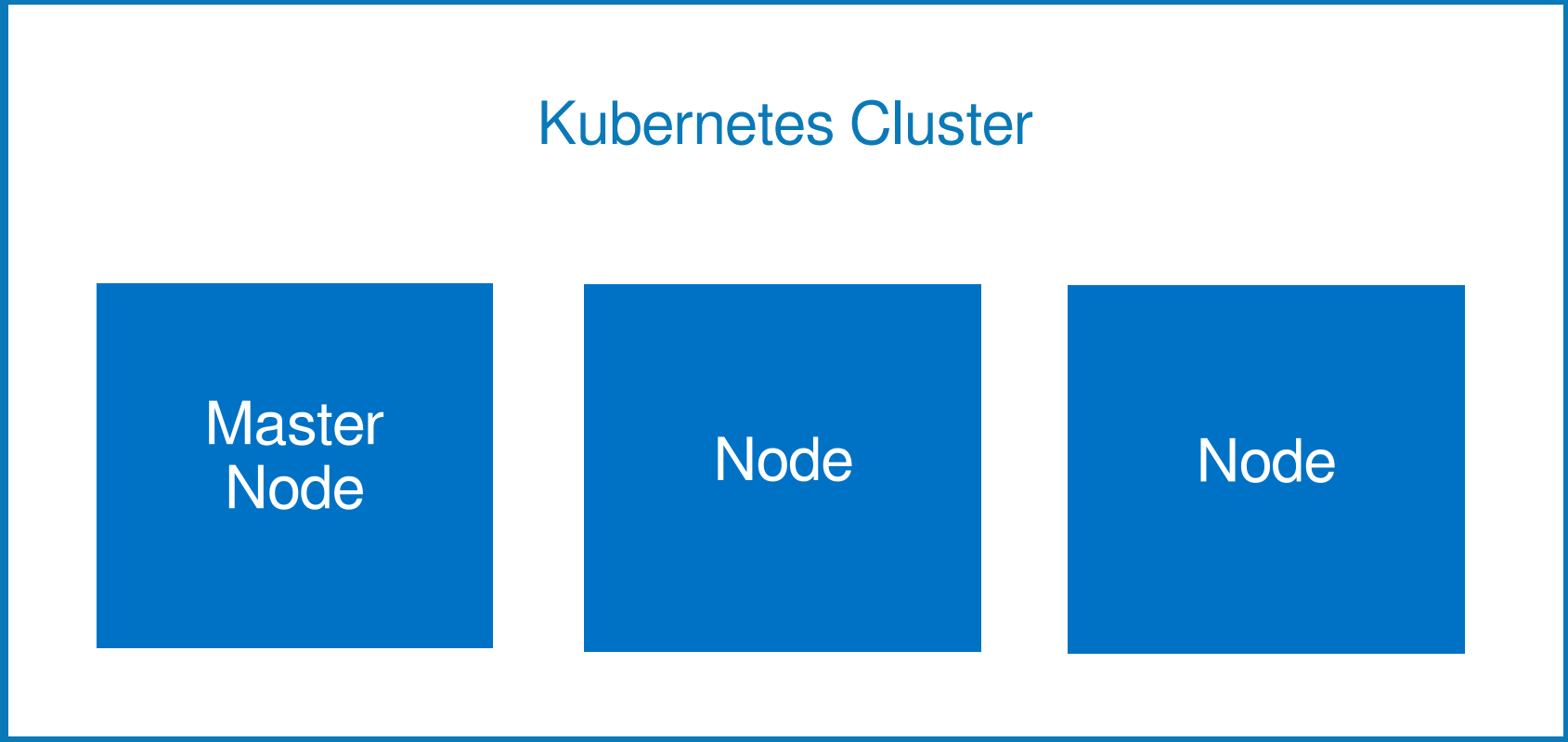
What's Kubernetes?

"Open-source system for automating deployment, scaling, and management of containerized applications" - Kubernetes website

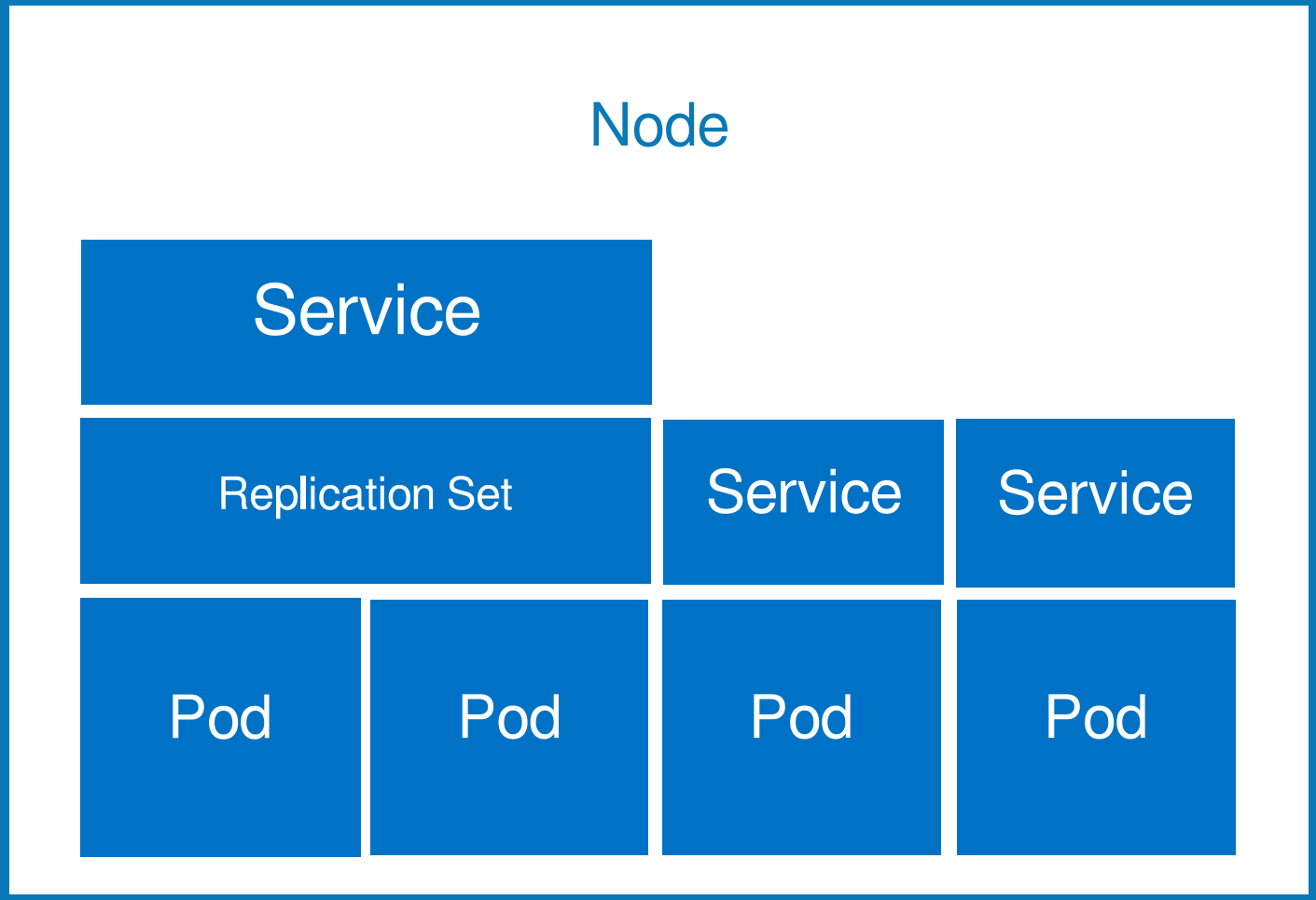
What Kubernetes can do?

- We can use Kubernetes to manage our containerized applications as well as automating deployment and scaling of the applications.
- Kubernetes can help us to update the deployed applications without downtime
- Many more! (90 mins won't be enough to cover them!)

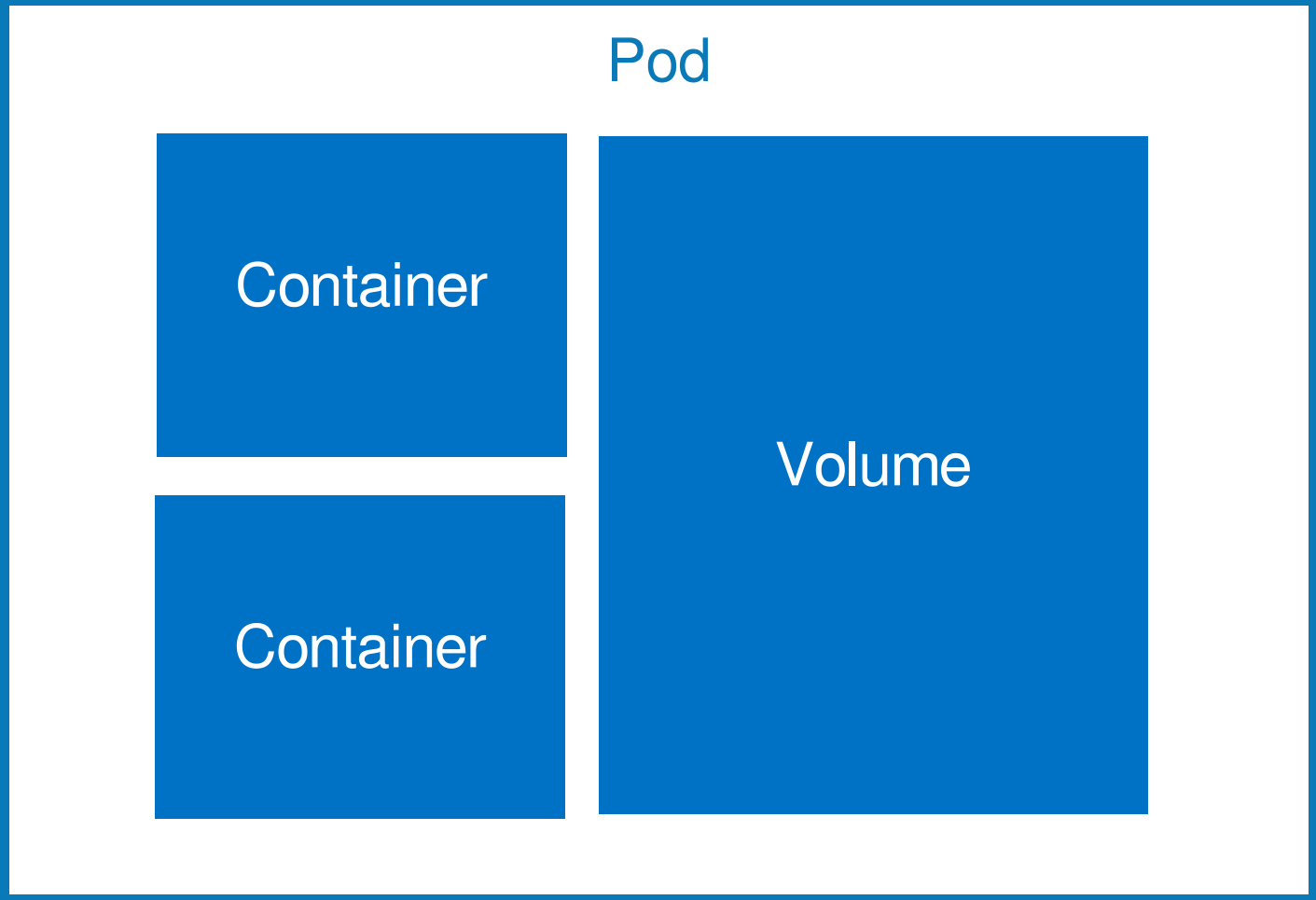
Kubernetes Components



Kubernetes Components



Kubernetes Components



Deploying on Kubernetes Cluster

- When we deploy our applications to the cluster, we describe the desired state of our application
  - Say we need to replicate a REST API application up to 4 replicas
  - Kubernetes will handle it and make sure the state of the deployment match our desired state
- Using *kubectl* CLI to manage objects in our cluster
  - We can provide full options in a single command line execution, or
  - We can describe our deployment using one or more YAML file(s)

Deploying on Kubernetes Cluster

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

Deploying on Kubernetes Cluster

```
$ kubectl create -f nginx.yaml
```

- Kubernetes will create a deployment with name nginx-deployment which has 3 replicas of pods using nginx image.

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-7b5dd5f6d7-jhtbs	0/1	ContainerCreating	0	12s
nginx-deployment-7b5dd5f6d7-ld825	0/1	ContainerCreating	0	12s
nginx-deployment-7b5dd5f6d7-tdjpm	0/1	ContainerCreating	0	12s

Scaling with Kubernetes

```
$ kubectl scale --replicas=4 deployment nginx-deployment

$ kubectl get pods

NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-7b5dd5f6d7-7zv9h   1/1       Running   0          22s
nginx-deployment-7b5dd5f6d7-jhtbs   1/1       Running   0          6m
nginx-deployment-7b5dd5f6d7-ld825   1/1       Running   0          6m
nginx-deployment-7b5dd5f6d7-tdjpm   1/1       Running   0          6m
```

The Problems with Kubernetes

- Kubernetes cluster setup is hard (at least for me).
- There's a long process of Kubernetes cluster setup before we can use tool like kubectl
  - In Kubernetes Cookbook, there is a dedicated chapter for setting up daemons required by Kubernetes cluster
  - You need at least three linux virtual machines / physical computers (1 master, 2 nodes) to install the daemons

Introducing Azure Container Service - AKS

Azure Container Service - AKS

- Uses Kubernetes for orchestrating Docker containers
- Makes your Kubernetes cluster setup to only few clicks away
- AKS does not require you to have expertise in orchestration
- AKS is still in **PREVIEW**
  - Some aspects are expected to change before GA

AKS - Pricing

- Users pay for resources they use
  - Cluster node (virtual machines) and their supporting resources
    - virtual networks
    - storage
    - etc
- Cluster management is **NOT** charged to users

What's in AKS Cluster?

- Network Security Group
- Route Table
- Virtual Machine(s)
  - With size DS1 v2 3.5 GB RAM 1 vCPU (resizable)
- Virtual Network
- Network Interface
- Disk
- Availability Set

Azure Container Service (AKS) in action

(demo)

Recap

- Deployment prior to container era is painful and prone to error
- Docker containers eliminates dependencies conflict and make applicaton more consistent regardless of what OS hosting them
- Managing and scaling containerized applications is hard, we need tools like Kubernetes or Docker Swarm to do that
- Setting up Kubernetes or Docker Swarm cluster on our own is even more harder
- Azure Container Services (AKS) relieves us from the responsibility of setting up Kubernetes cluster and ease the deployment, scaling and managing cluster so we can focus on the application development.

Any Questions?

Thank You!