docker for DevOps Engineers 2019

by Siarhei Beliakou

part 4

AGENDA

- Containerized Application Design Principles
- Docker Swarm Mode

Containerized Application Design Principles

- Cloud Native Infrastructure
- Cloud Native Application
- 12 Factor Application

Cloud Native Infrastructure

Cloud native infrastructure is infrastructure that is hidden behind useful abstractions, controlled by APIs, managed by software, and has the purpose of running applications. Running infrastructure with these traits gives rise to a new pattern for managing that infrastructure in a scalable, efficient way.

This includes data centers, operating systems, deployment pipelines, configuration management, and any system or software needed to support the life cycle of applications.

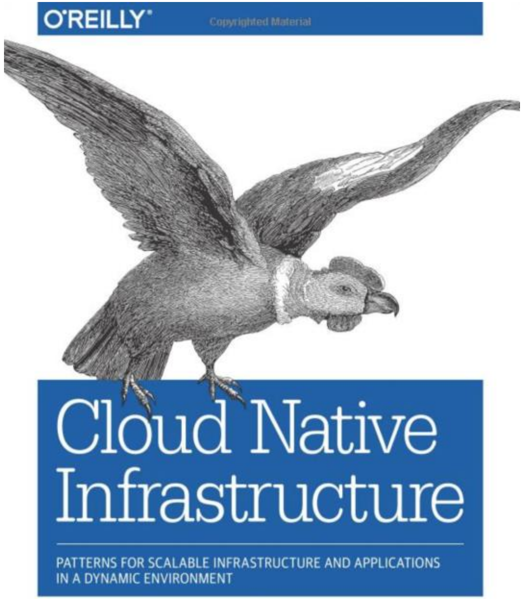
Cloud native infrastructure is a requirement to effectively run cloud native applications. Without the right design and practices to manage infrastructure, even the best cloud native application can go to waste.

Abstractions should always allow the consumer to “move up the stack” and not reimplement the lower layers.

Cloud Native Application

A cloud native application is engineered to run on a platform and is designed for:

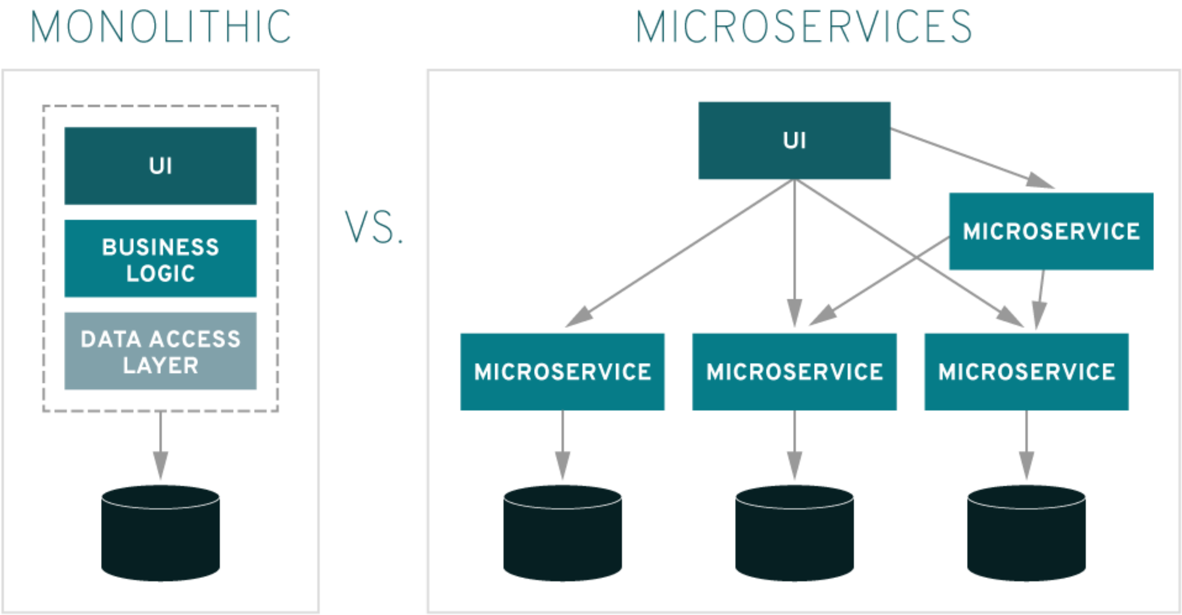
- **Resiliency** - embraces failures instead of trying to prevent them; it takes advantage of the dynamic nature of running on a platform.
- **Agility** - allows for fast deployments and quick iterations.
- **Operability** - adds control of application life cycles from inside the application instead of relying on external processes and monitors. Observability provides information to answer questions about application state.


O'REILLY
Copyrighted Material
Cloud Native Infrastructure
PATTERNS FOR SCALABLE INFRASTRUCTURE AND APPLICATIONS IN A DYNAMIC ENVIRONMENT

The following are common ways to implement the desired characteristics of a cloud native application:

- Microservices
- Health reporting
- Telemetry data (Requests Rate/Errors/Response Duration)
- Resiliency (design for failure, graceful degradation)
- Declarative, not reactive

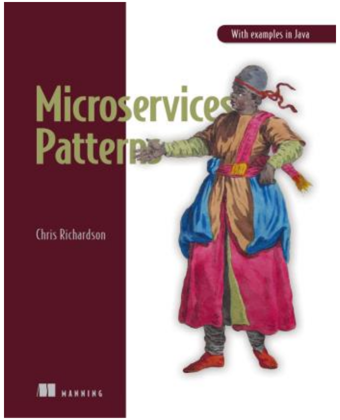
Microservices



Microservices is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities.

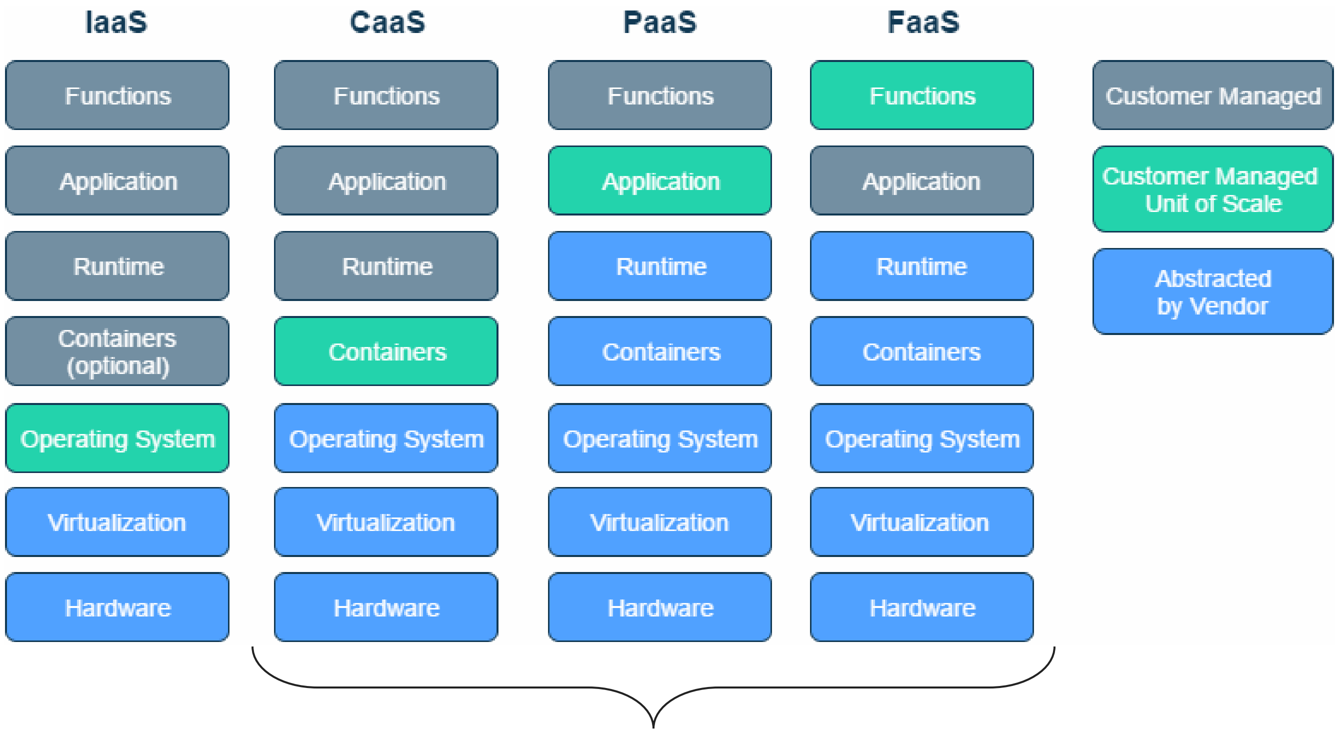
https://microservices.io/



https://microservices.io/patterns/monolithic.html

https://microservices.io/patterns/microservices.html

IaaS/CaaS/PaaS/FaaS



IaaS	CaaS	PaaS	FaaS
Functions	Functions	Functions	Functions
Application	Application	Application	Application
Runtime	Runtime	Runtime	Runtime
Containers (optional)	Containers	Containers	Containers
Operating System	Operating System	Operating System	Operating System
Virtualization	Virtualization	Virtualization	Virtualization
Hardware	Hardware	Hardware	Hardware

Customer Managed

Customer Managed Unit of Scale

Abstracted by Vendor

Cloud Native Platform Scope

The 12 Factor Application

The twelve-factor app is a methodology for building ***software-as-a-service*** applications

1. **Codebase** - One codebase tracked in revision control, many deploys
2. **Dependencies** - Explicitly declare and isolate dependencies
3. **Config** - Store config in the environment
4. **Backing services** - Treat backing services as attached resources
5. **Build, release, run** - Strictly separate build and run stages
6. **Processes** - Execute the app as one or more stateless processes
7. **Port binding** - Export services via port binding
8. **Concurrency** - Scale out via the process model
9. **Disposability** - Maximize robustness with fast startup and graceful shutdown
10. **Dev/prod parity** - Keep development, staging, and production as similar as possible
11. **Logs** - Treat logs as event streams
12. **Admin processes** - Run admin/management tasks as one-off processes

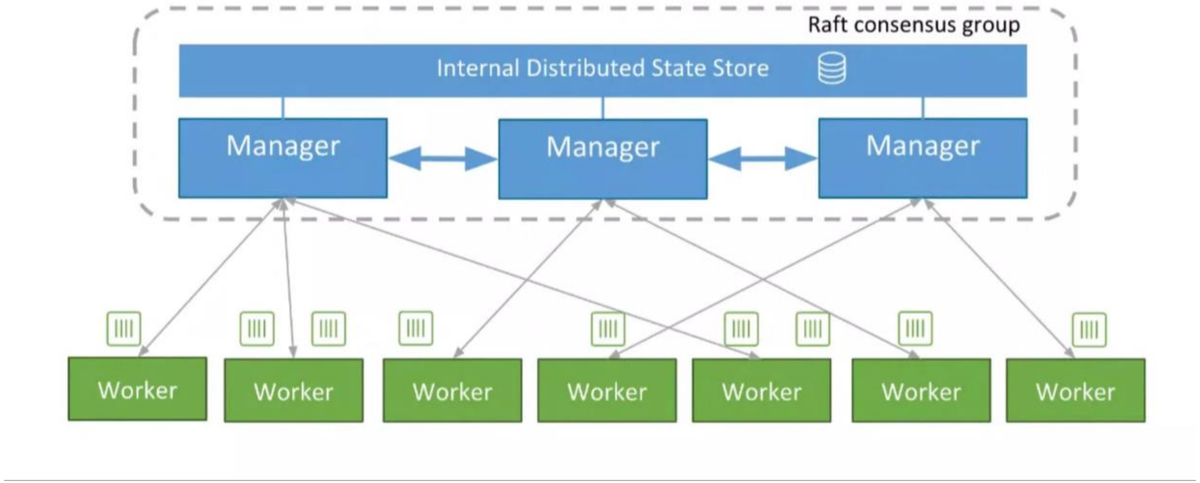
https://12factor.net/

https://github.com/docker/labs/tree/master/12factor

Docker SWARM 101

- Swarm Architecture
- Swarm Services
- Swarm Stacks

Docker SWARM Architecture



Key Components:

- Swarm Manager
- Swarm Node (worker)
- Scheduler
- Discovery

https://docs.docker.com/engine/swarm/
https://docs.docker.com/engine/swarm/key-concepts/
https://docs.docker.com/engine/swarm/admin_guide/
https://success.docker.com/article/networking#swarmnativeservicediscovery

Docker SWARM Communication Ports

The network ports required for a Docker Swarm to function properly are:

- TCP port **2376** for secure Docker client communication. This port is required for Docker Machine to work. Docker Machine is used to orchestrate Docker hosts.
- TCP port **2377**. This port is used for communication between the nodes of a Docker Swarm or cluster. It only needs to be opened on manager nodes.
- TCP and UDP port **7946** for communication among nodes (container network discovery).
- UDP port **4789** for overlay network traffic (container ingress networking).

```
$ firewall-cmd --add-port=2376/tcp --permanent
$ firewall-cmd --add-port=2377/tcp --permanent
$ firewall-cmd --add-port=7946/tcp --permanent
$ firewall-cmd --add-port=7946/udp --permanent
$ firewall-cmd --add-port=4789/udp --permanent

$ firewall-cmd --reload
$ systemctl restart docker
```

https://www.digitalocean.com/community/tutorials/how-to-configure-the-linux-firewall-for-docker-swarm-on-centos-7

SWARM Init

Initialize Manager:

```
# docker swarm init --force-new-cluster --advertise-addr 192.168.56.16
Swarm initialized: current node (mjwv0xeam6ebnne3soswbn01h) is now a manager.
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-590c5aoirtpiypl5t... vc5m4lu0b5l5kqh65nday 192.168.56.16:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Get token for joining to Manager:

```
# docker swarm join-token worker -q
```

Add worker node, and check the cluster:

```
$ docker node ls
ID                            HOSTNAME    STATUS               AVAILABILITY        MANAGER STATUS      ENGINE
VERSION
mjwv0xeam6ebnne3soswbn01h  *  manager     Ready                Active              Leader              18.06.1-ce
4r0b4gnz7dj9th1ro5ilppemr     worker      Ready                Active                                  18.06.1-ce
```

Create Overlay Network:

```
$ docker network create -d overlay skynet
```

SWARM Services

```
# docker service create \
  --name echo \
  --network skynet \
  --replicas 2 \
  -p 80:80 \
  sbeliakou/httpd-echo
wxqek2mpwvjvy0vlaefg7bi6u
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
```

```
# docker service ls
ID                NAME      MODE            REPLICAS    IMAGE                         PORTS
wxqek2mpwvjv      echo      replicated      2/2         sbeliakou/httpd-echo:latest   *:80->80/tcp
```

```
# docker service ps echo
ID             NAME     IMAGE                         NODE      DESIRED STATE   CURRENT STATE            ERROR    PORTS
z5hgske7ceym   echo.1   sbeliakou/httpd-echo:latest   manager   Running         Running 55 seconds ago
jbbknrx4n3hj   echo.2   sbeliakou/httpd-echo:latest   worker    Running         Running 55 seconds ago
```

```
# docker service logs echo
```

SWARM Services

```
# docker service inspect --pretty echo
ID:                    wxqek2mpwvjvy0vlaefg7bi6u
Name:                  echo
Service Mode:          Replicated
 Replicas: 2
Placement:
UpdateConfig:
 Parallelism:          1
 ...
 Update order:         stop-first
RollbackConfig:
 Parallelism:          1
 ...
 Rollback order:       stop-first
ContainerSpec:
 Image:                sbeliakou/httpd-echo:latest@sha256:6e5724ce1b630...d5a052890f4a4f8f27d3c1da492be46c7c67
 Init:                 false
Resources:
Networks: skynet
Endpoint Mode:         vip
Ports:
 PublishedPort = 80
  Protocol = tcp
  TargetPort = 80
  PublishMode = ingress
```

SWARM Services

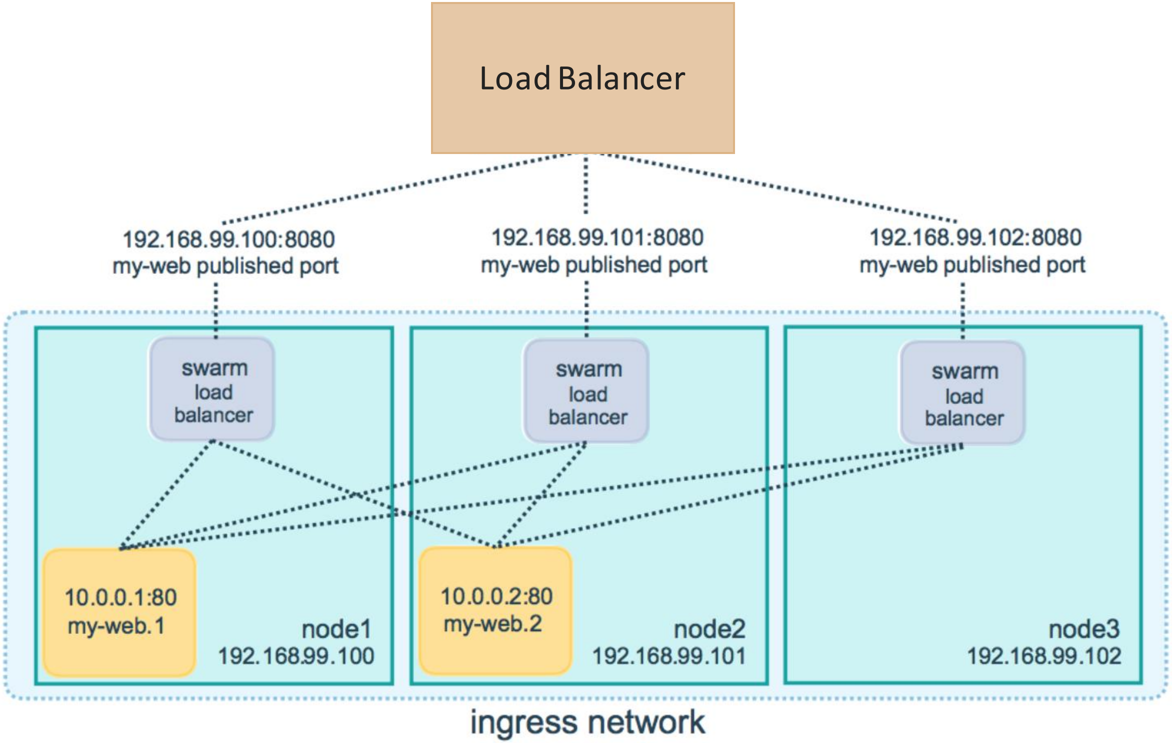
```
# docker service scale echo=5
echo scaled to 5
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged
```

```
# docker service ps echo
ID            NAME    IMAGE                          NODE       DESIRED STATE  CURRENT STATE           ERROR   PORTS
z5hgske7ceym  echo.1  sbeliakou/httpd-echo:latest   manager    Running        Running 3 minutes ago
jbbknrx4n3hj  echo.2  sbeliakou/httpd-echo:latest   worker     Running        Running 3 minutes ago
j3mo0flxod9r  echo.3  sbeliakou/httpd-echo:latest   manager    Running        Running 8 seconds ago
2tjkcv1eg4l6  echo.4  sbeliakou/httpd-echo:latest   worker     Running        Running 10 seconds ago
574s2uh9thqt  echo.5  sbeliakou/httpd-echo:latest   manager    Running        Running 7 seconds ago
```

```
# curl 192.168.56.16
Request was processed on 05f330567dda
# curl 192.168.56.16
Request was processed on e596c60c08ec
...
```

```
# docker service rm echo
```

SWARM Ingress Network & Routing Mesh



https://docs.docker.com/engine/swarm/ingress/

Deployment with Docker-Compose

docker-compose.yml

```
version: "3.6"

services:
  web:
    image: sbeliakou/httpd-echo
    ports:
    - 80:80
    deploy:
      replicas: 2
```

https://docs.docker.com/compose/compose-file/
https://docs.docker.com/docker-cloud/apps/stack-yaml-reference/

Deploy Stack with Compose:

```
# docker stack deploy --compose-file web-stack.yml mystack
Creating network mystack_default
Creating service mystack_web
```

List stacks:

```
# docker stack ls
NAME                SERVICES            ORCHESTRATOR
mystack             1                   Swarm
```

SWARM Stacks

List the services in the stack

```
# docker stack services mystack
ID            NAME         MODE        REPLICAS   IMAGE                          PORTS
p6o7jjzb5vk9  mystack_web  replicated  2/2        sbeliakou/httpd-echo:latest    *:81->80/tcp
```

Get logs from the service:

```
# docker service logs mystack
mystack_web.1.lfiybkzqnsza@ip-10-136-1-6.eu-west-1.compute.internal    | 10.255.0.4 - - [27/Nov/2018:11:36:53 +0000]
"GET / HTTP/1.1" 200 211 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/69.0.3497.100 Safari/537.36 OPR/56.0.3051.104"
```

List the tasks in the stack

```
# docker stack ps mystack
ID            NAME           IMAGE                        NODE         DESIRED STATE  CURRENT STATE
nib4xefqrs0z  mystack_web.1  sbeliakou/httpd-echo:latest  docker-host  Running        Running 12 minutes ago
py8tzqqs48f2  mystack_web.2  sbeliakou/httpd-echo:latest  docker-host  Running        Running 12 minutes ago
```

Remove the stack:

```
# docker stack rm mystack
Removing service mystack_web
Removing network mystack_default
```

P.S.

○ **What have we touched?**

○ **What's else?**

What We Have Covered

- Containerized Application Design Principles
- Docker Architecture
- Docker Installation and Configuration
- Building Docker Images
- Running Docker Containers
- Docker Volumes
- Docker Networks
- Docker Logging System
- Linux Kernel Namespaces and CGroups
- Linux Kernel Capabilities
- Docker-in-Docker Concept
- Docker Remote Access
- Docker SWARM 101
- Monitoring Containers
- Containers Security

Do you think this is enough for running Docker in Production?

Check your Docker Daemon Configuration:

```
# curl https://raw.githubusercontent.com/docker/docker/master/contrib/check-
config.sh | bash -
```

Docker Security: https://docs.docker.com/engine/security/security/

Use Trusted Registries and Images: https://docs.docker.com/engine/security/trust/content_trust/

Keep containers alive during daemon downtime (*/etc/docker/daemon.json*):

```
{
    "live-restore": true
}
```

live-restore daemon configuration is **incompatible with swarm mode**

```
# docker info | grep Live
Live Restore Enabled: true
```

And This is Only Origins …

https://docs.docker.com/config/labels-custom-metadata/
https://docs.docker.com/engine/security/userns-remap/
https://docs.docker.com/config/pruning/
https://docs.docker.com/config/daemon/
https://docs.docker.com/config/containers/logging/configure/
https://docs.docker.com/config/containers/logging/log_tags/
https://docs.docker.com/config/containers/logging/fluentd/
https://docs.docker.com/registry/recipes/mirror/
https://docs.docker.com/notary/getting_started/
…

That's it for this training!

Thank you for your attention!

Siarhei Beliakou,

2019