Docker Swarm Vs Kubernets

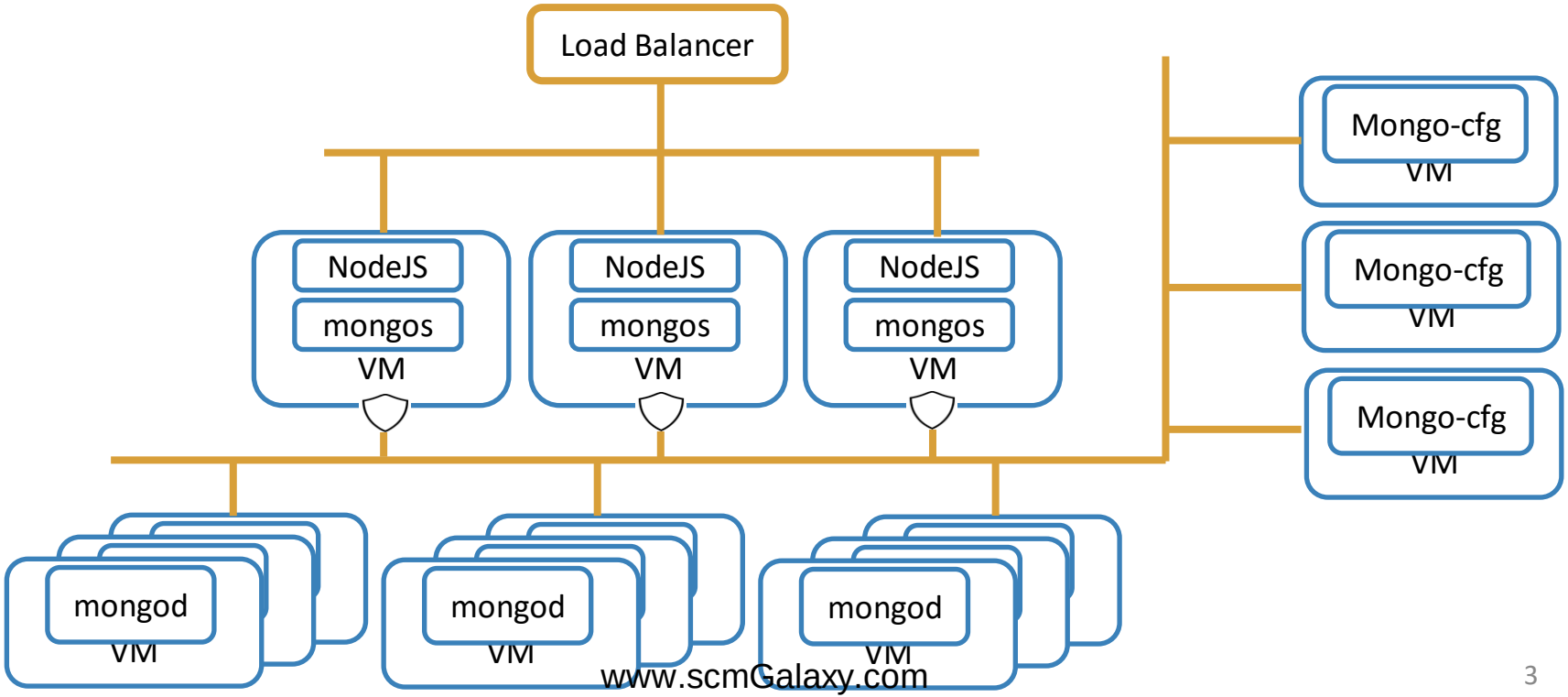
Rajesh Kumar

DevOps Architect

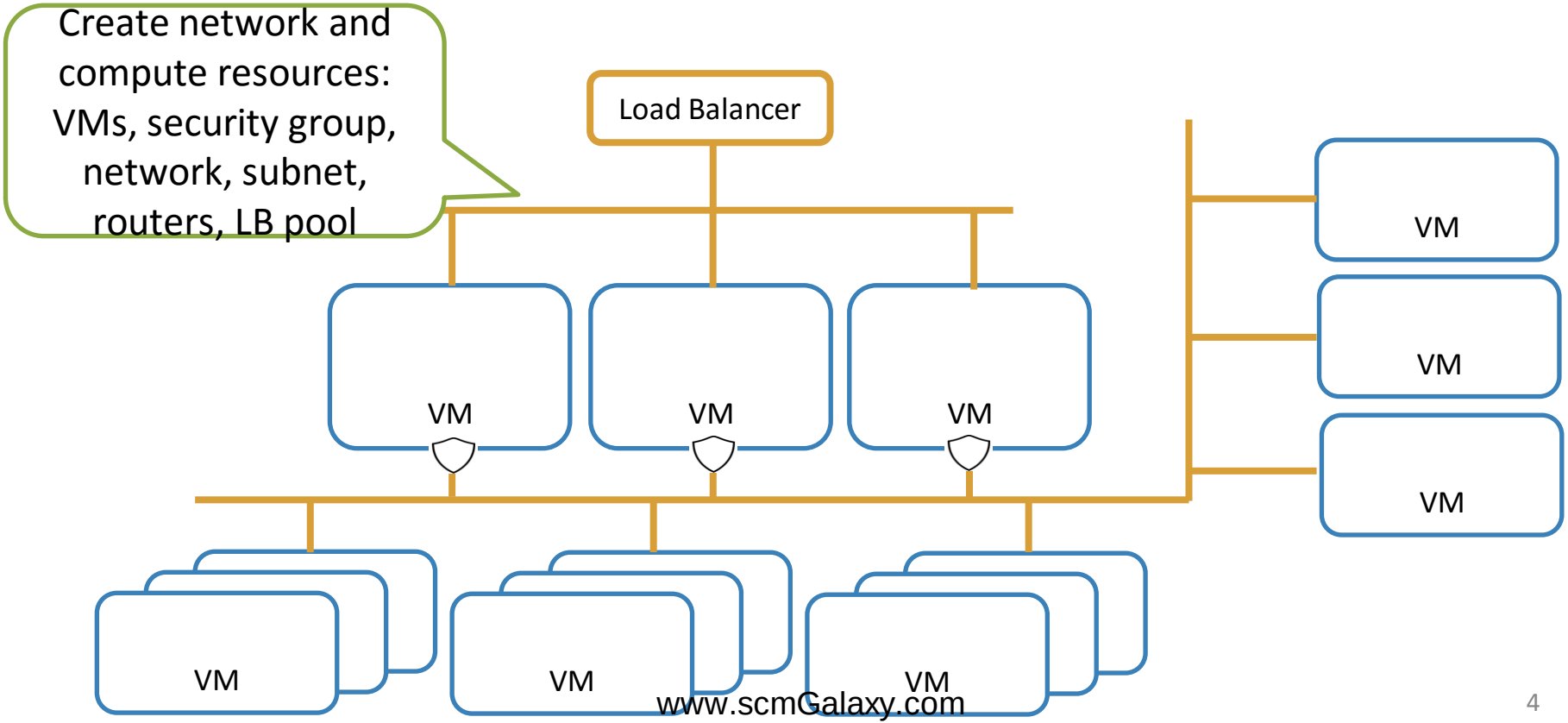
@RajeshKumarIN | www.RajeshKumar.xyz

Orchestration Process

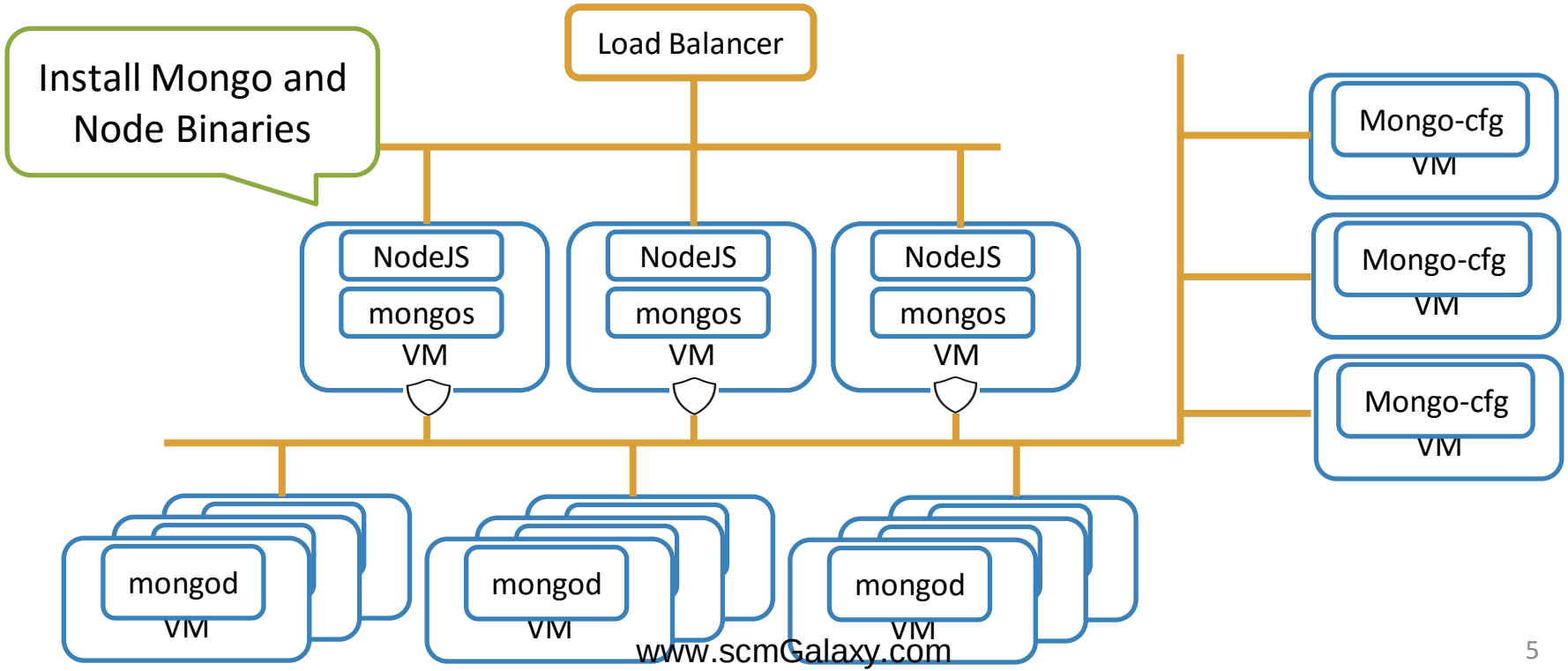
The Test Application



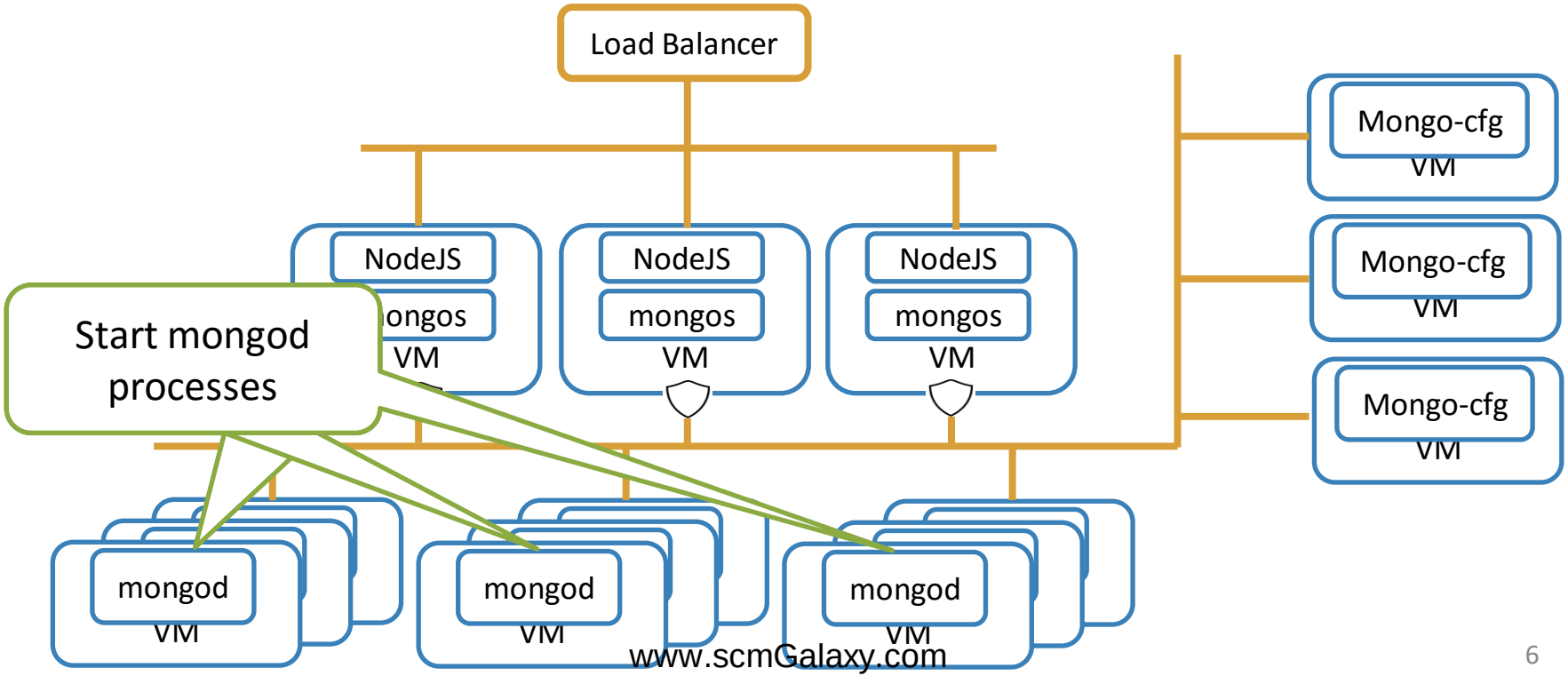
1 Orchestration Process - Setup



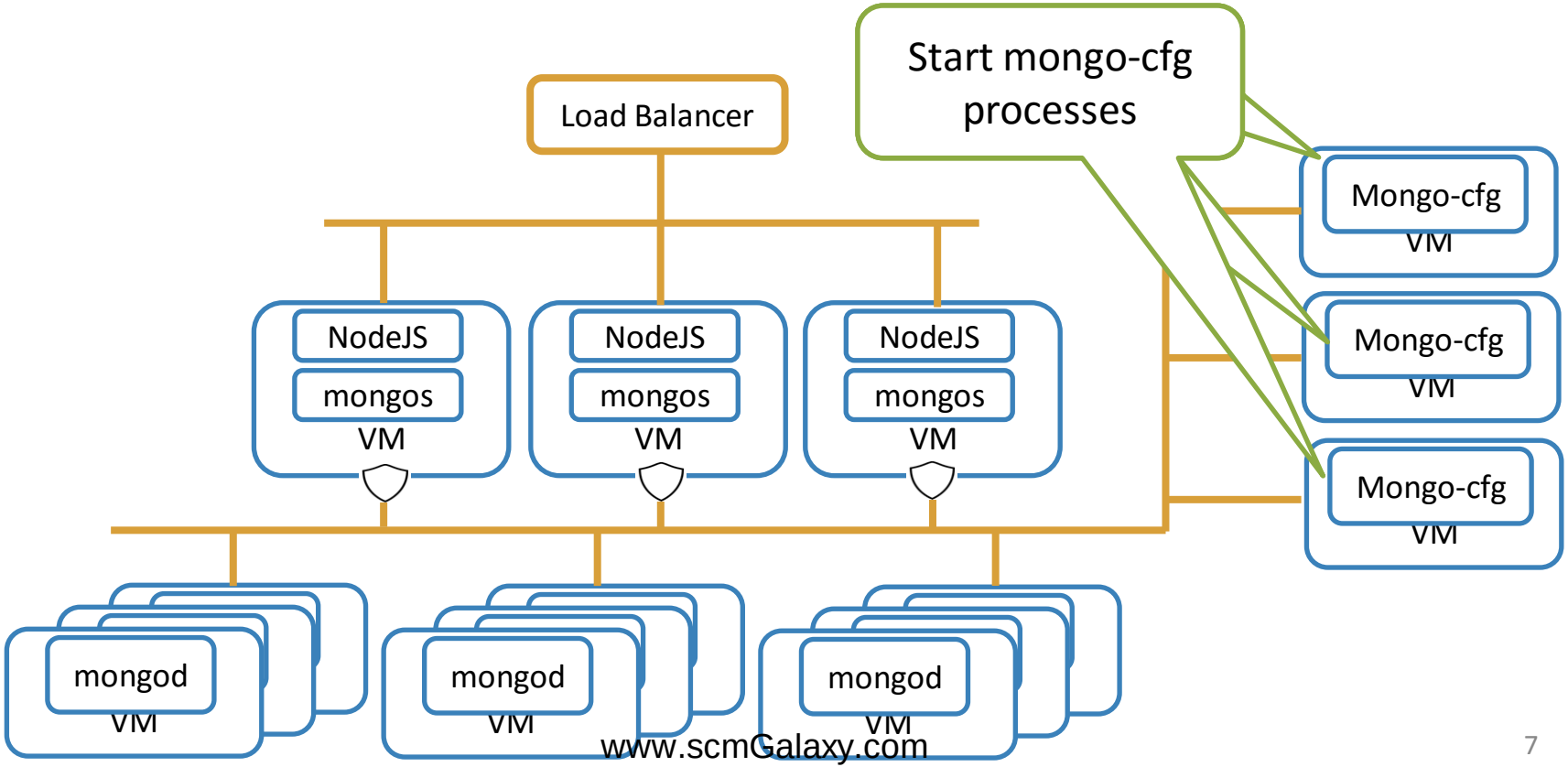
2 Orchestration Process - Setup



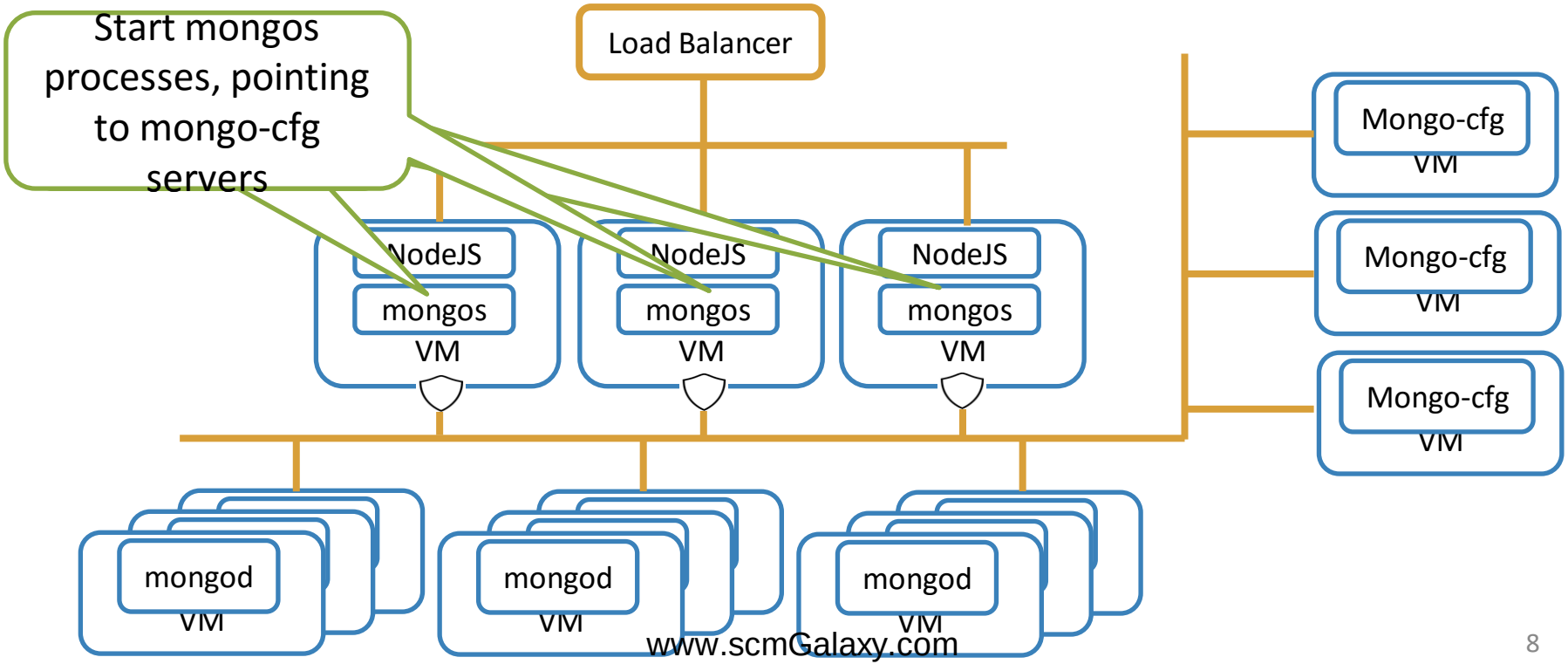
3 Orchestration Process - Setup



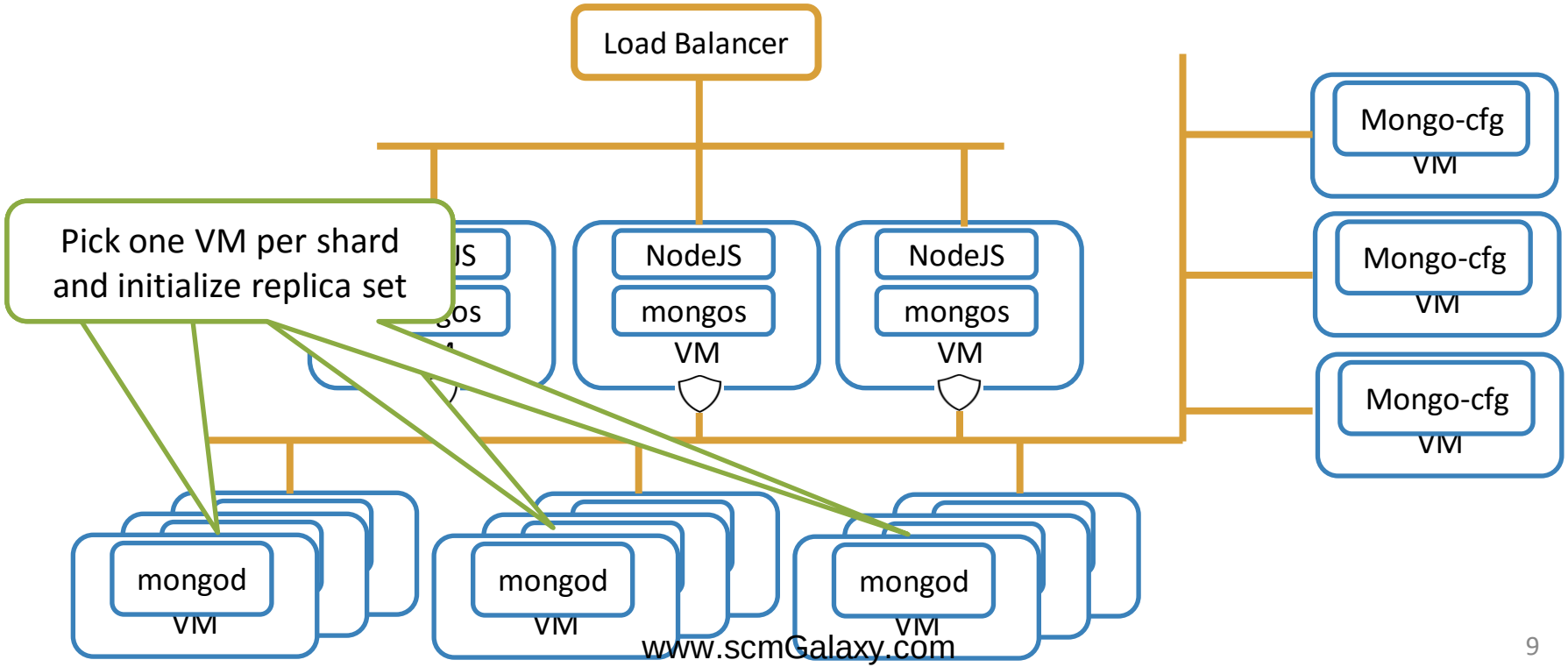
4 Orchestration Process - Setup



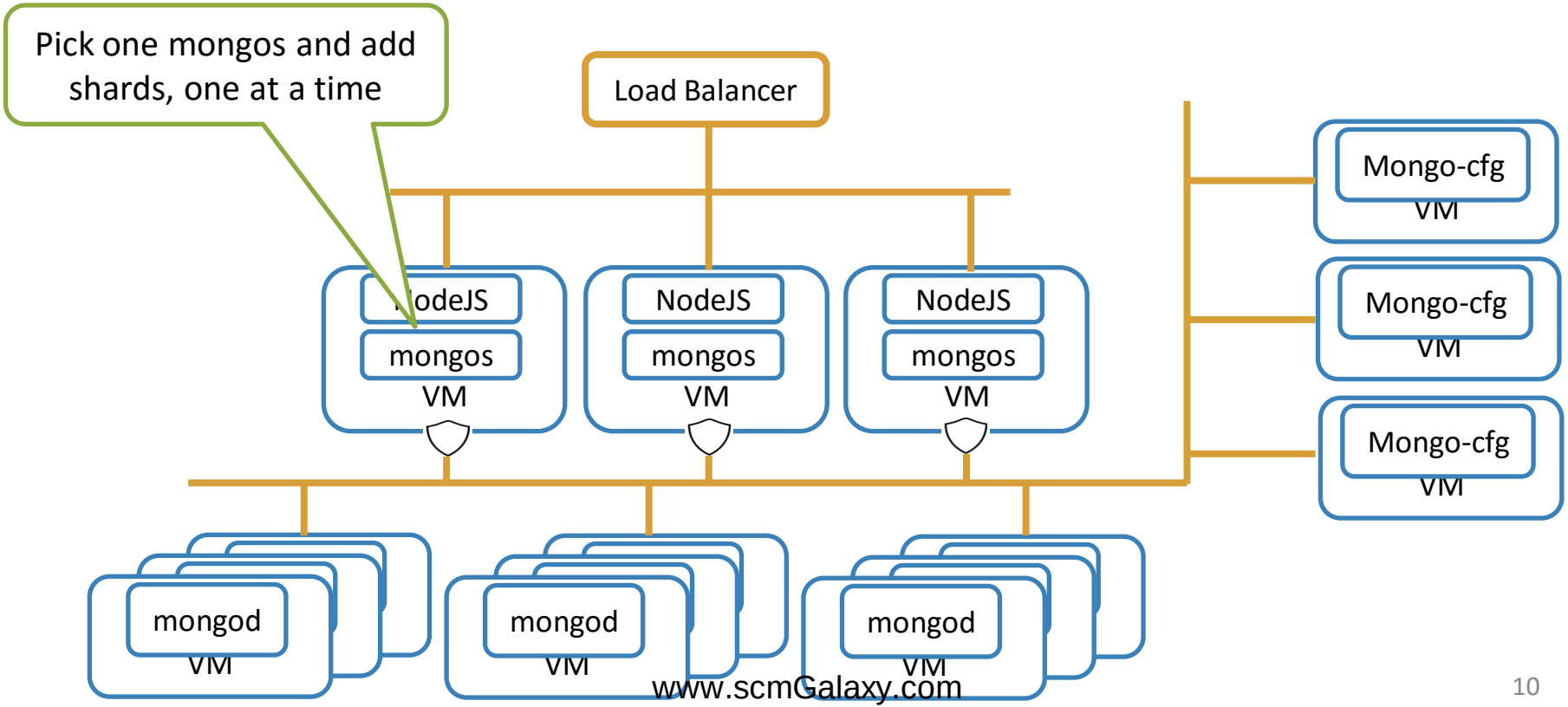
5 Orchestration Process - Setup



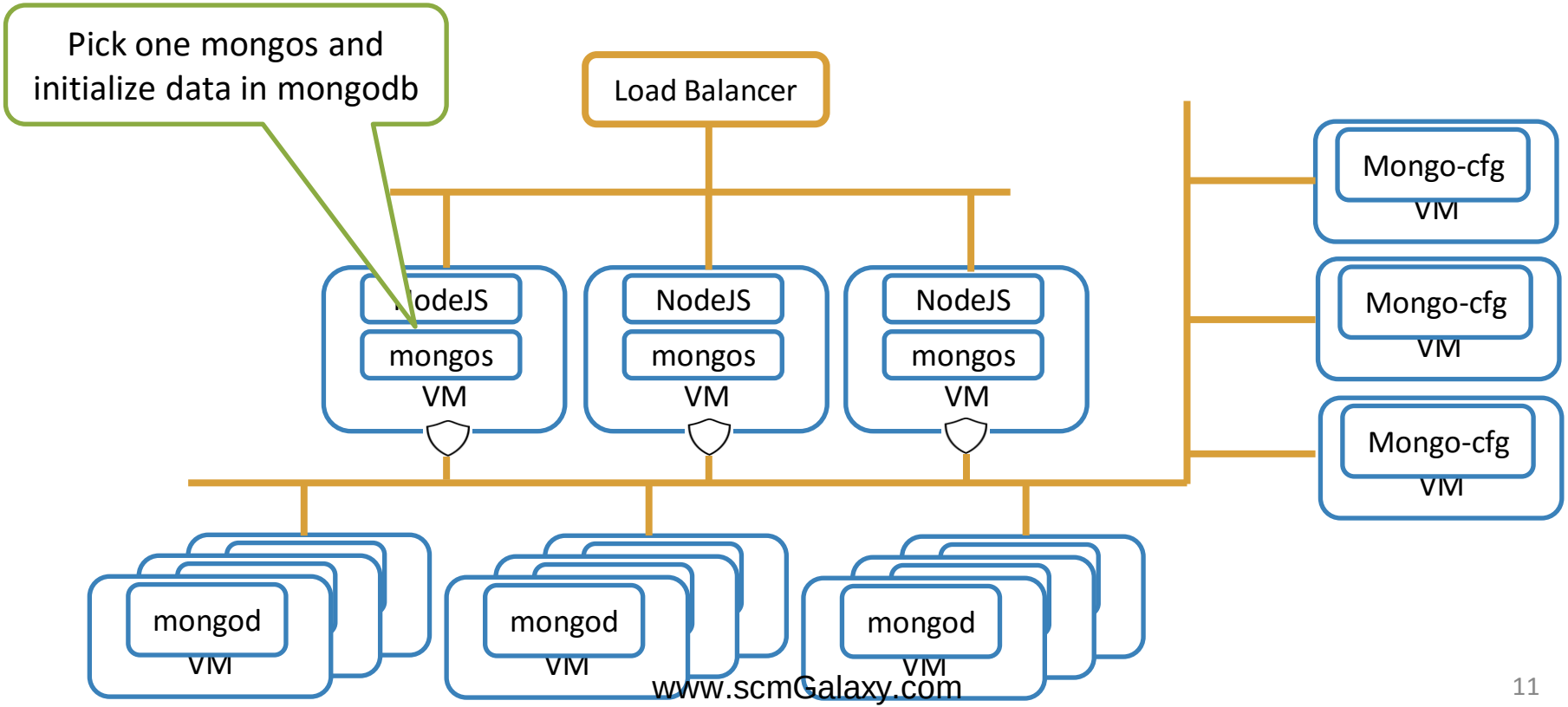
6 Orchestration Process - Setup



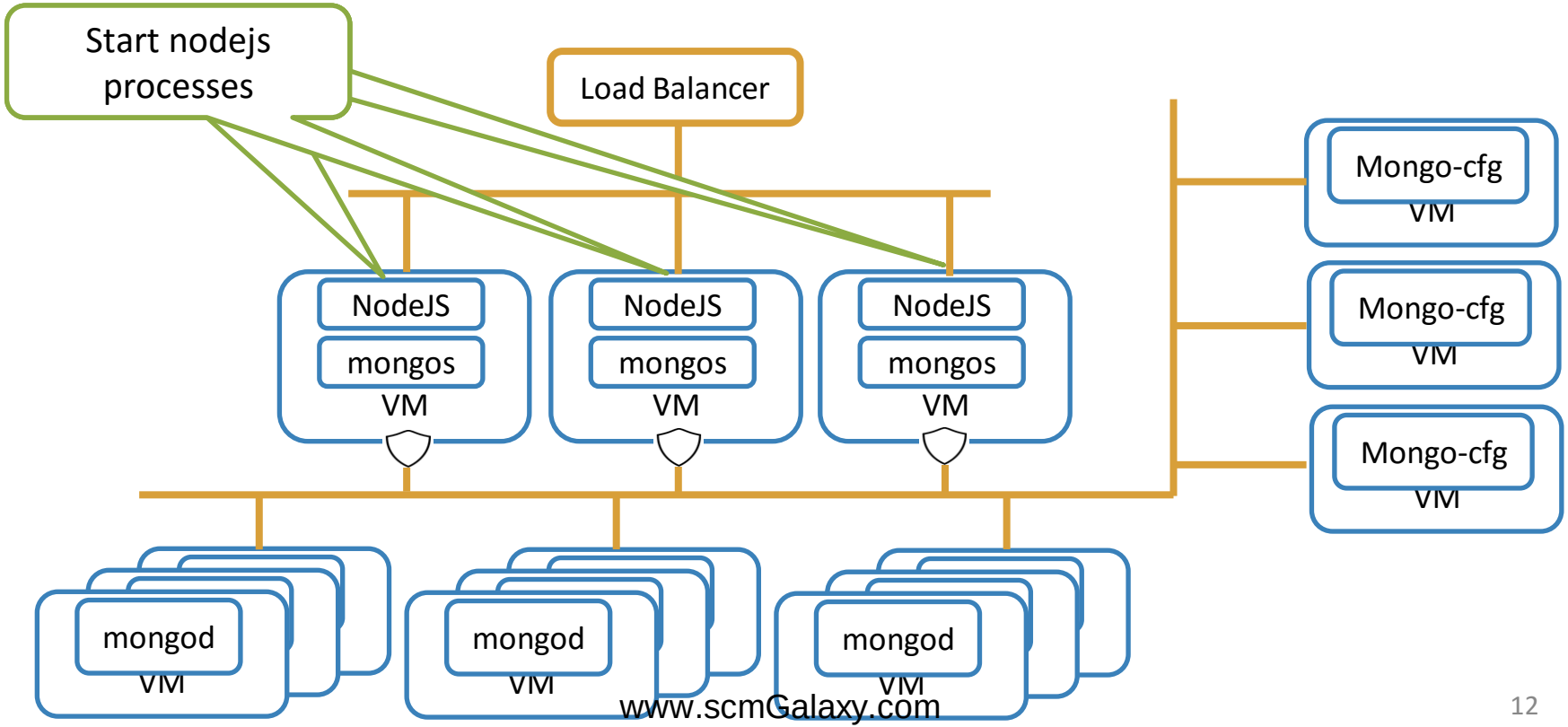
7 Orchestration Process - Setup



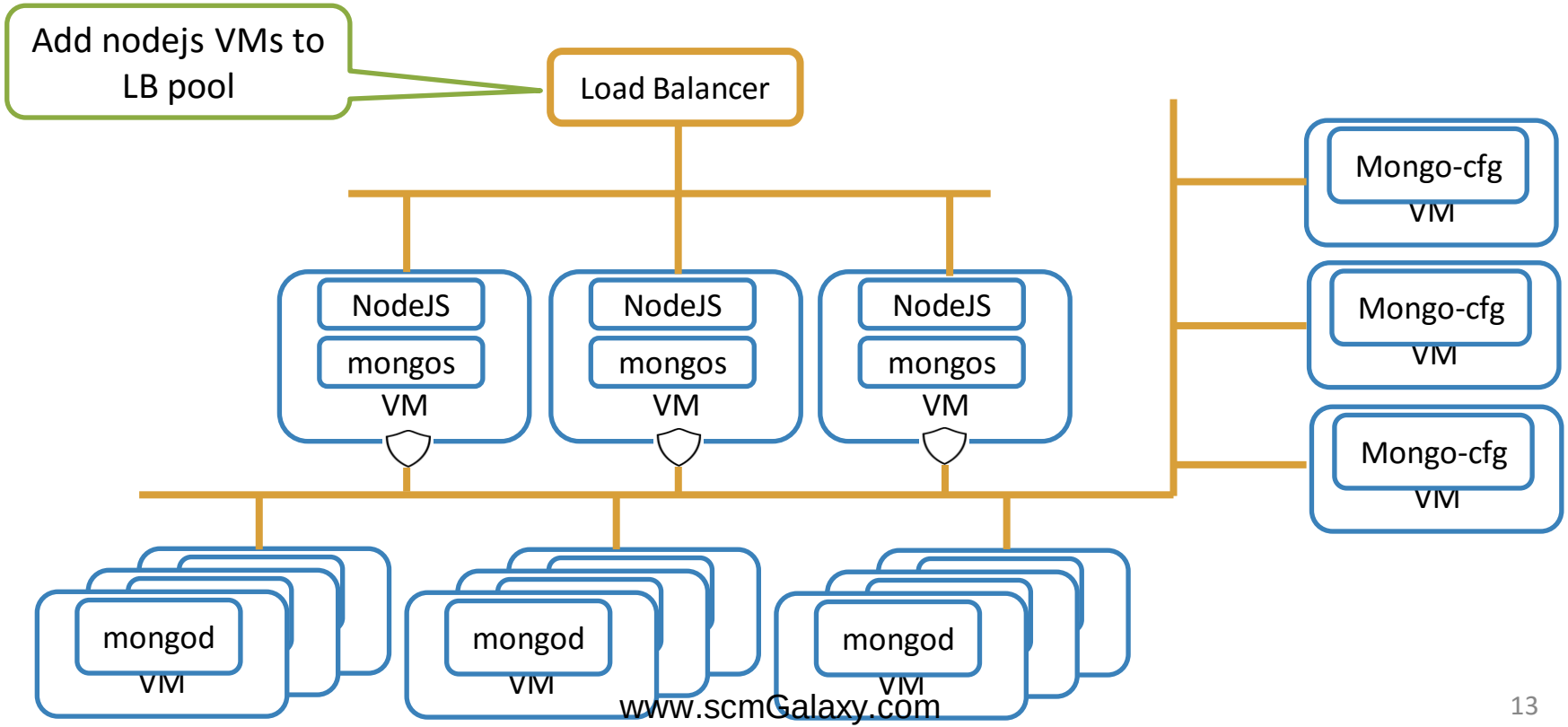
8 Orchestration Process - Setup



9 Orchestration Process - Setup



10 Orchestration Process - Setup

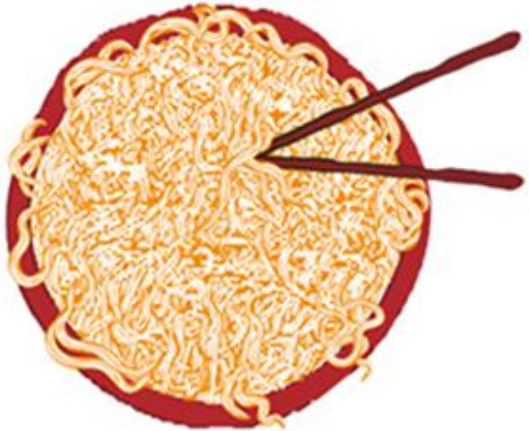



Docker Orchestrators:
Swarm vs. Kubernetes
Jelastic, PaaS & CaaS solution for Hosting companies

Microservices

1990s and earlier

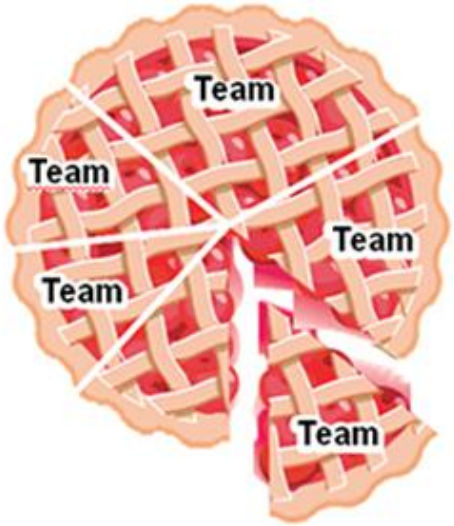
Pre-SOA (monolithic)
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

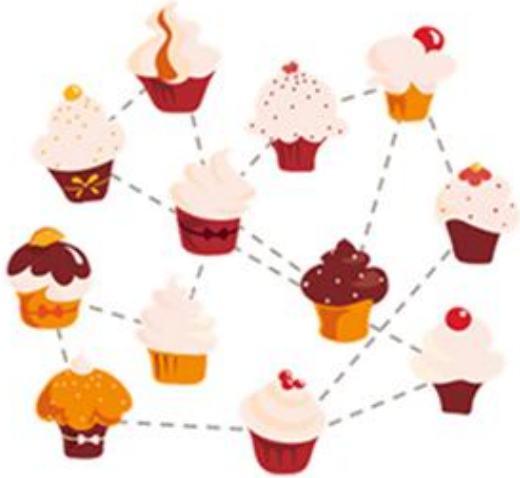
Traditional SOA
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

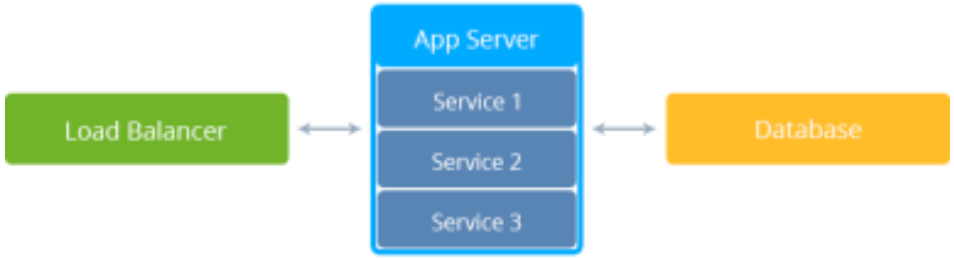
2010s

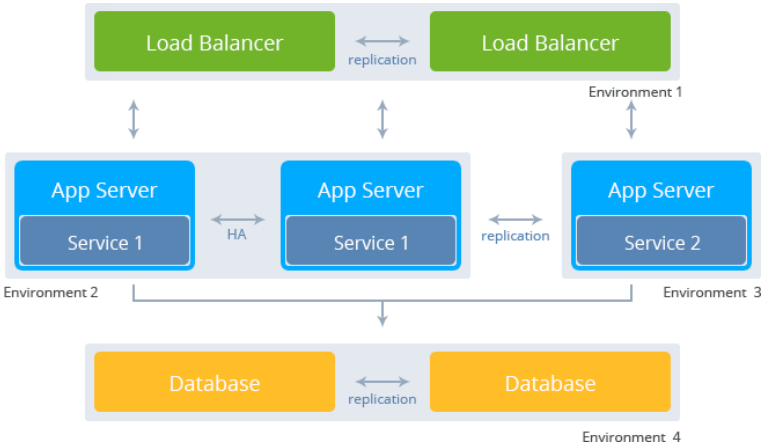
Microservices
Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible

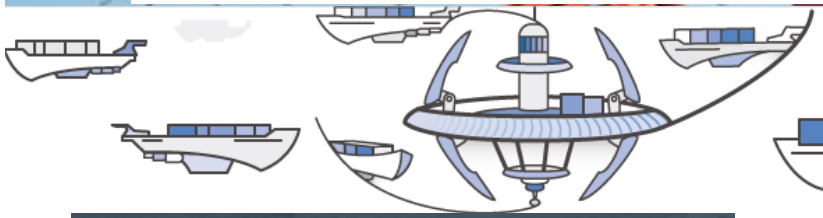
Microservices vs. Monolith





Container orchestrator

Amazon EC2 Container Service



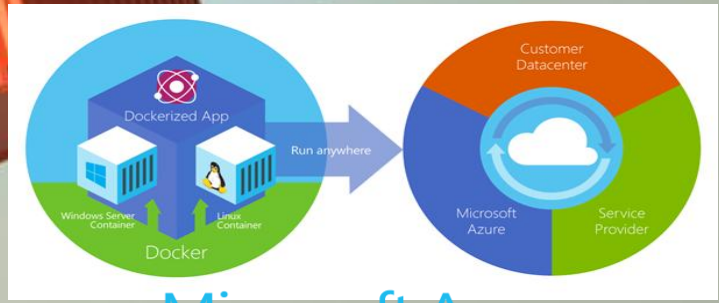
TECTONIC
by CoreOS

CoreOS Stack + Kubernetes

The Best Platform for Linux Containers

SALTSTACK

IBM and SaltStack for Docker App Container Orchestration and Networking at Scale



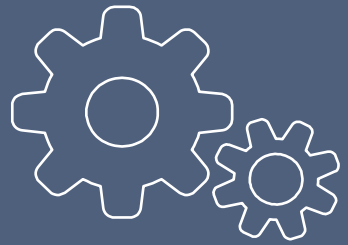
Microsoft Azure

Containers might be a real salvation for the developers

But this is not so simple

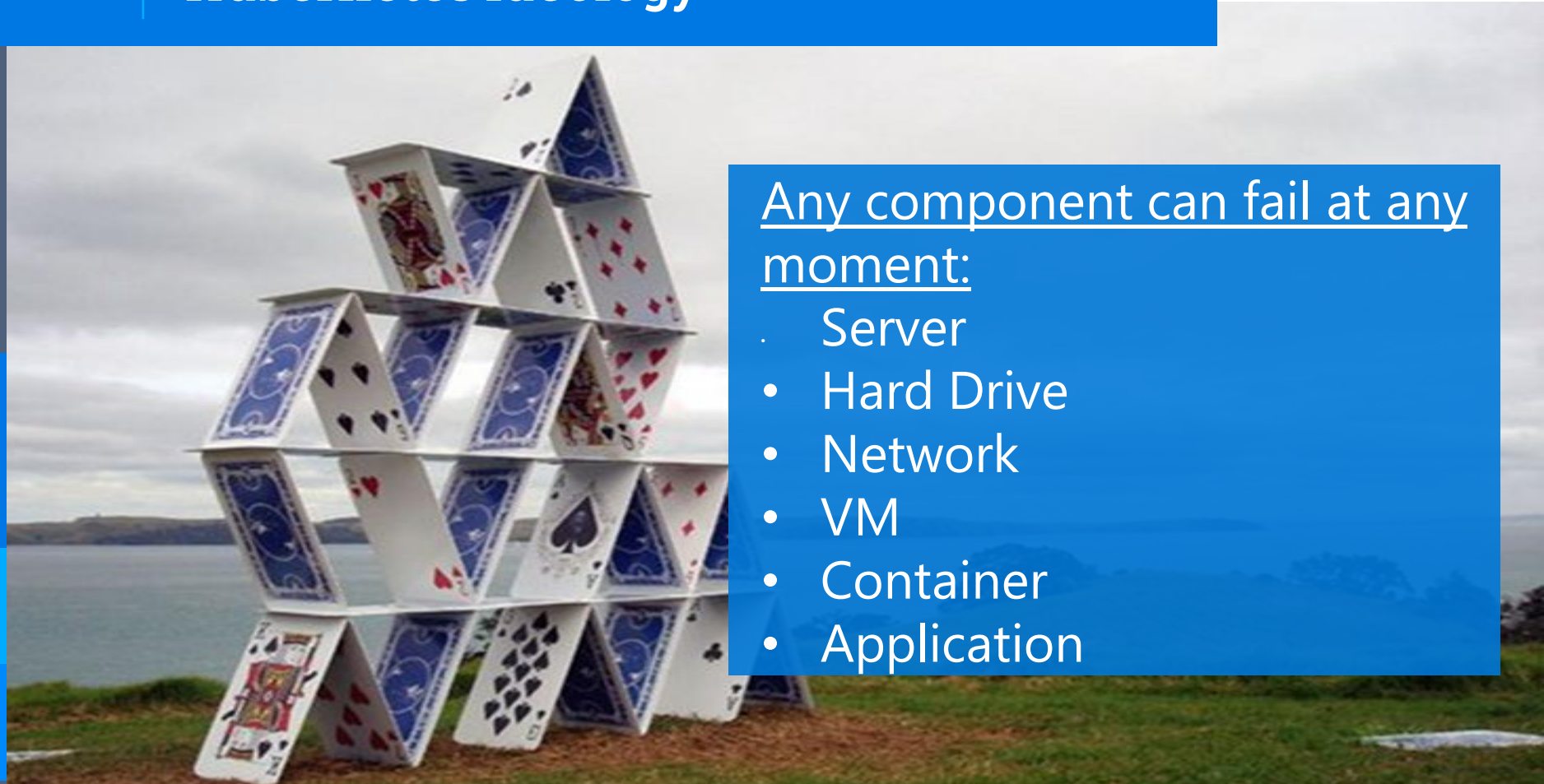
Yet now to simple

How would you build 1000 Node application being not able to login to its admin panel at all?



Kubernetes

Kubernetes ideology



Any component can fail at any moment:

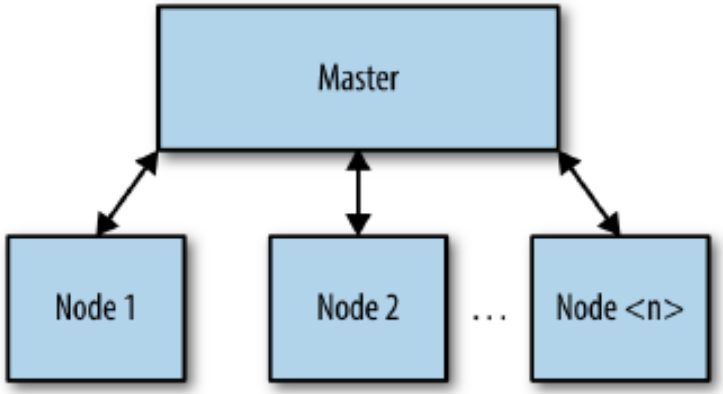
- Server
- Hard Drive
- Network
- VM
- Container
- Application

What opportunities Kubernetes gives to you?

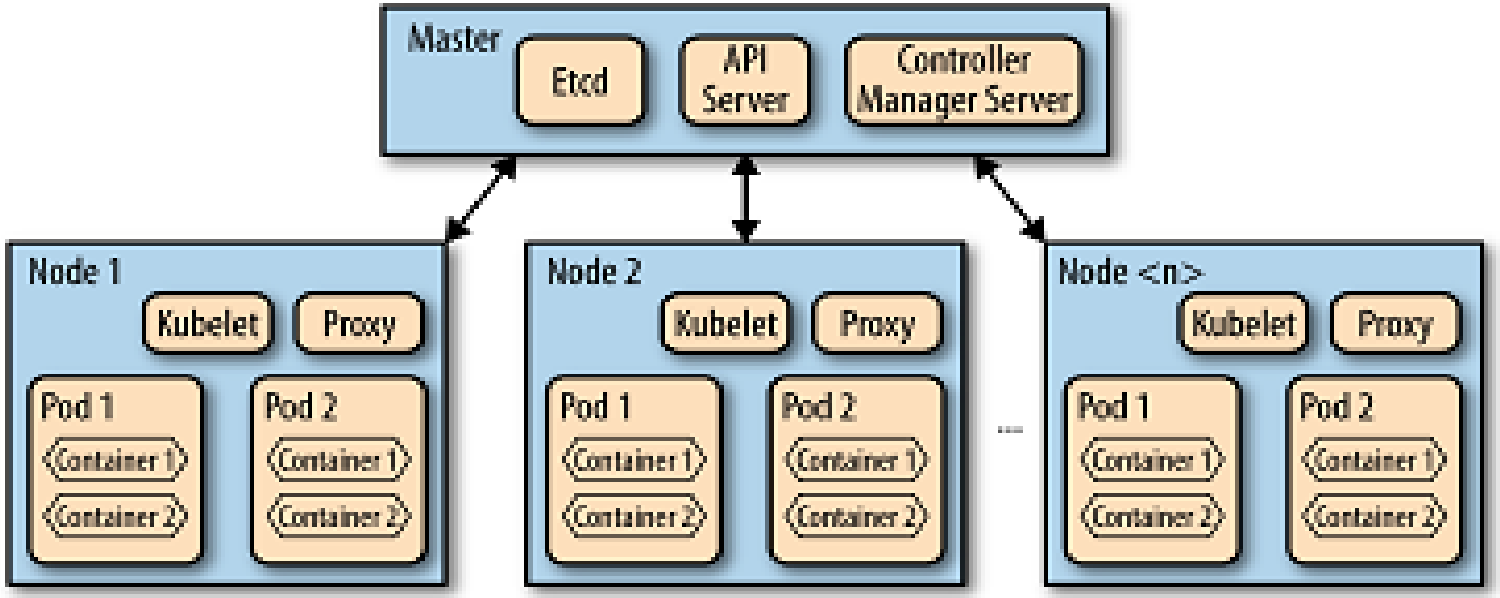
1. You can declaratively describe the application configuration(YAML)
2. Each group of components (Pod) can be named by (Label)
3. You describe how groups of components have to be duplicated, for example a service should be running in 5 copies
4. Kubernetes deploys specified configuration on the available infrastructure (Nodes)
5. Kubernetes ensures that the current configuration of the application is always consistent with the reference
6. There are built-in health checks, on the basis of assessment of applications health and replacement of corrupted Pod with new one
7. As a result, your application always has needed amount of running instances

Kubernetes architecture

1. Master – Orchestrator
2. Nodes – Servers, used for users' workloads

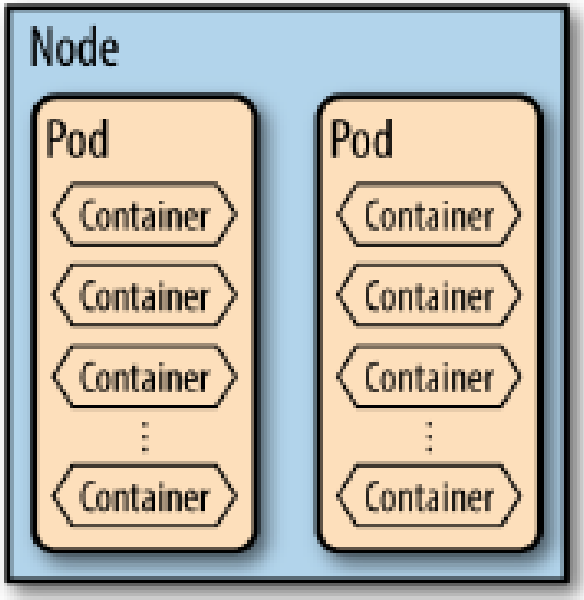


Kubernetes architecture



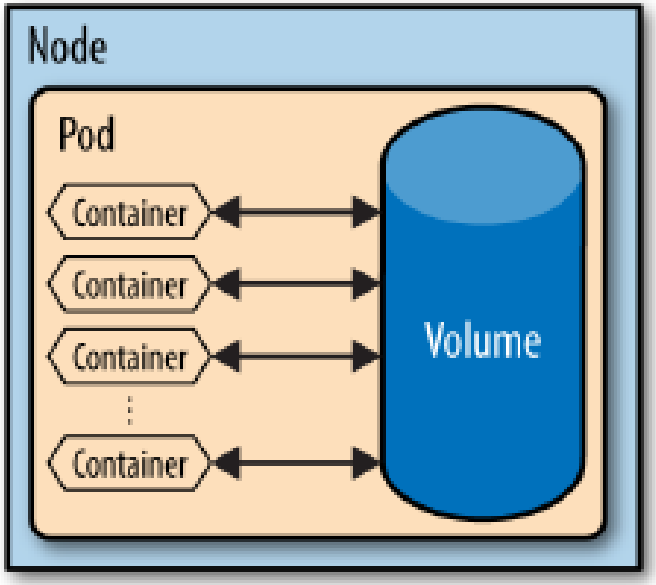
Nodes

1. Each node consists of a set of Pods
2. Each Pod consists of a set of linked containers.
3. Pod is a scaling unit in Kubernetes instead of the container
4. Kubernetes includes built-in algorithms for Anti-affinity



Volumes

1. Empty Dir
2. NFS
3. GCEPersistentDisk
4. awsElasticBlockStore
5. Glusterfs
6. Iscsi
7. Rbd
8. Secrets



Labels

1. Your configuration layout. KEY/VALUE
2. “**labels**”:{
   “**tier**”: “frontend”
   “**application.awesome-game/environment**”: “production”
}

A **label** is basically an arbitrary tag that can be placed on the above work units to mark them as a part of a group.

Label selector

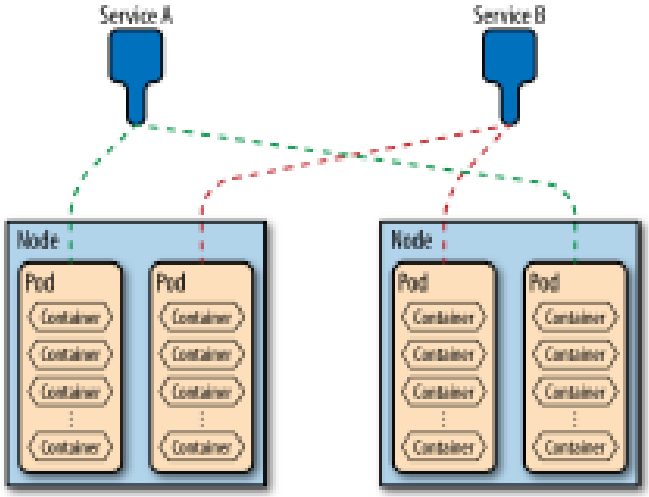
1. **Query mechanism to Labels**
2. tier **!=** frontend, game **=** super-shooter-2
3. environment **in** (production, qa)
   tier **notin** (frontend, backend)
   partition

Replication controller

1. Copies of single Pod are called “the replicas”
2. Replication Controller monitors the keeping of the needed level of the Pod replication
3. Replication Controller provides automatic failover
4. RP allows label changing for specified Pod thus eliminating it from replicating or implementing ZDT Rolling Updates
5. Scaling of the application may be done only manually via shift for a specific number of specific Pod replicas

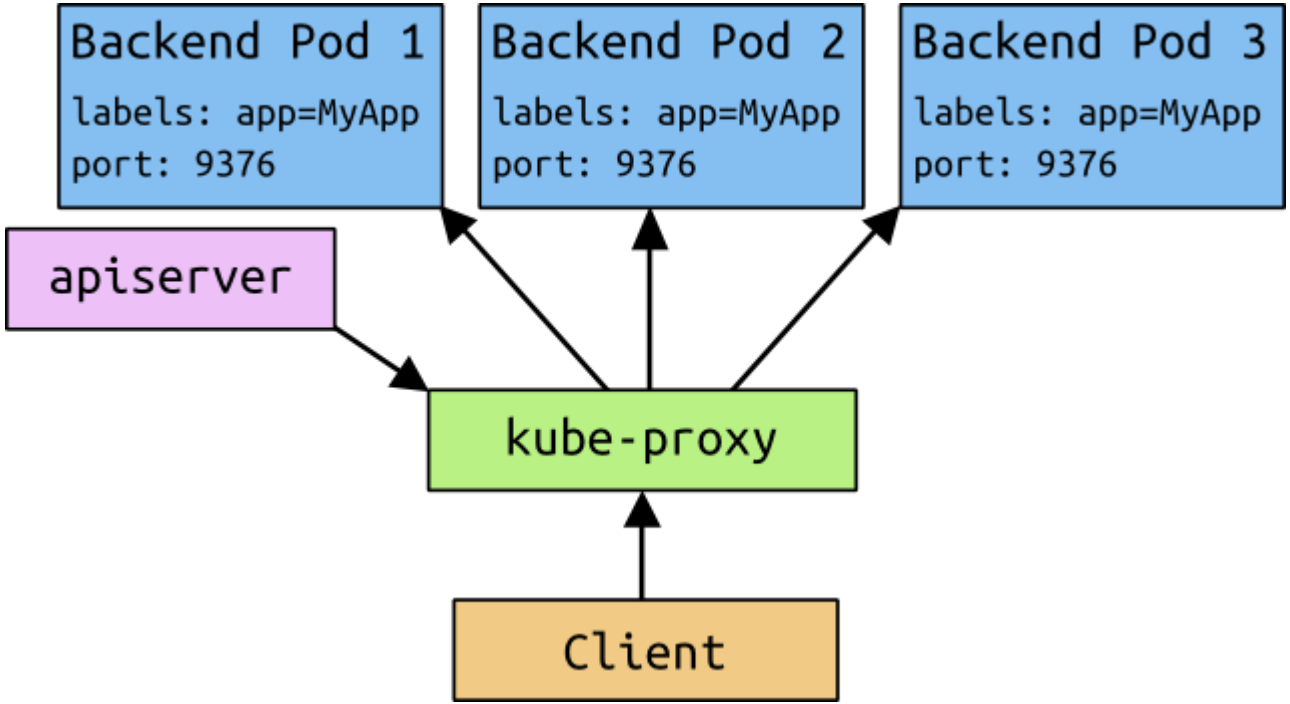
Services

1. Reverse-proxy
2. Services have external IP
3. Simple Round-Robin balancing



An endpoint that provides load balancing across a replicated group of pods

Services


Backend Pod 1
labels: app=MyApp
port: 9376
Backend Pod 2
labels: app=MyApp
port: 9376
Backend Pod 3
labels: app=MyApp
port: 9376
apiserver
kube-proxy
Client

Service discovery

- Pod environment variables appear on Node
- Cluster DNS- Special Pod type
  - Etcd –Configuration storage
  - SkyDns – DNS-server reads data from Etcd
  - Kube2sky –pushes the latest information from Kubernetes Master to Etcd

In a microservices application, the set of running service instances changes dynamically. Instances have dynamically assigned network locations. Consequently, in order for a client to make a request to a service it must use a service-discovery mechanism.

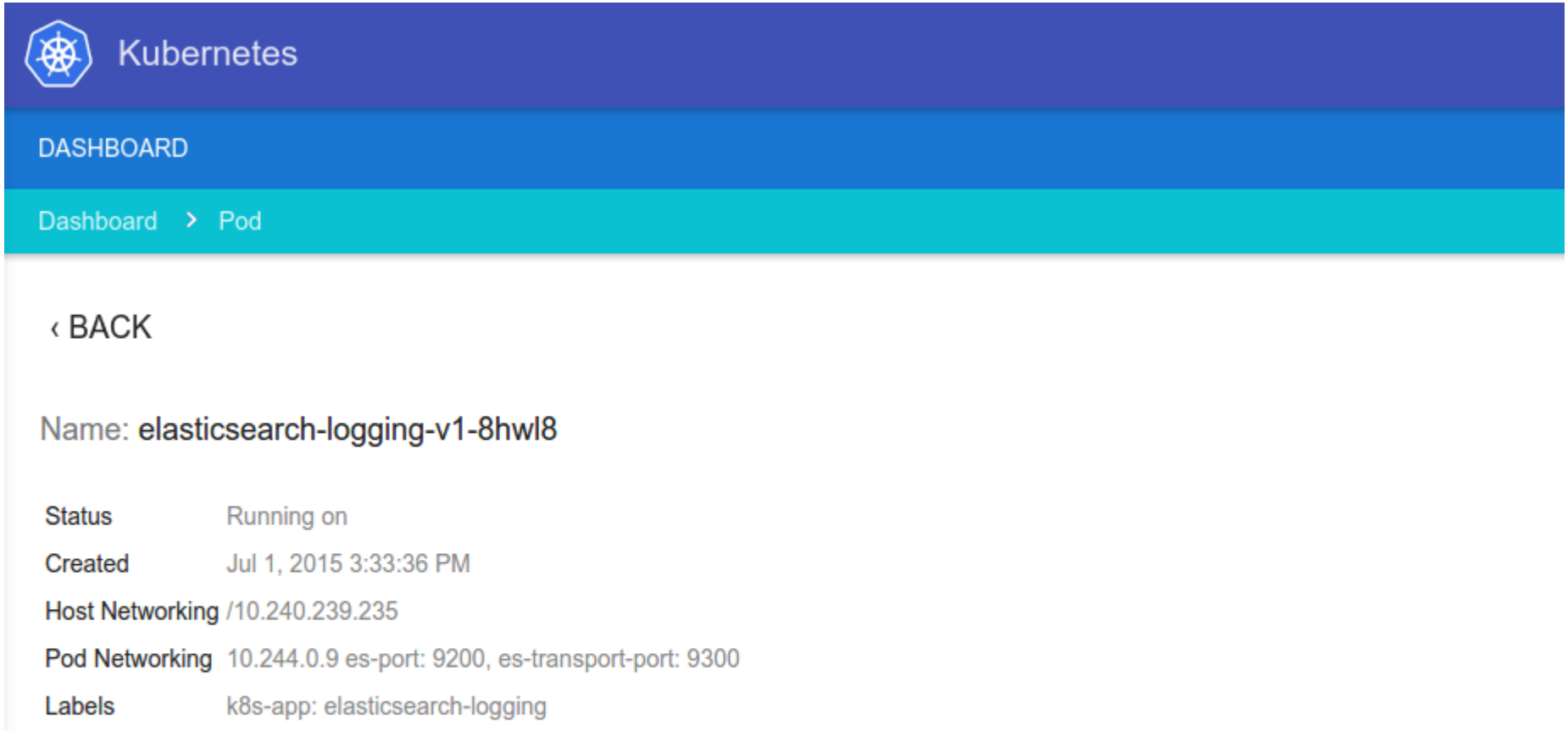
Good Reading - https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/

Health checks

- TCP Socket
- HTTP GET
- Container Exec

```
livenessProbe:
    enabled: true
    type: http
        initialDelaySeconds: 30
    httpGet:
        path: /_status/healthz
        port: 8080
```

System Dashboards



How to deploy?

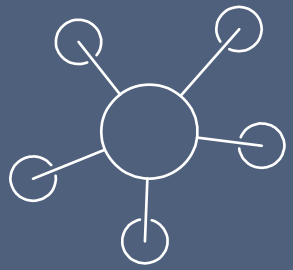
- Local (Docker-based)
- Vagrant
- Local (No VM)
- Hosted Solution:
  - Google Container Engine
  - AWS
  - Azure
  - Mesosphere
  - OpenStack Magnum/Murano

Kubernetes advantages

1. Kubernetes ensures that your application will always have the required number of running instances
2. Perfect for Rolling updates
3. Not so good for Stateful applications
4. There are some issues with the application auto scalability

Kubernetes limitations

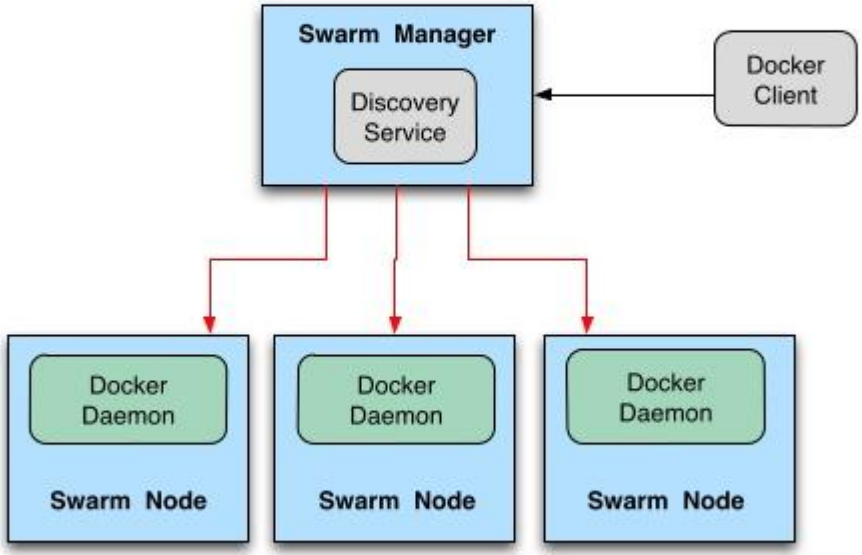
1. Complicated manual cluster deployment
2. Complicated automatic horizontal scaling set up
3. Most of the actions have to be performed in CLI
4. Orchestration logic is hidden deep in the Kubernetes
5. Container is not a control unit



Docker Swarm

Docker Swarm

Architecture Diagram



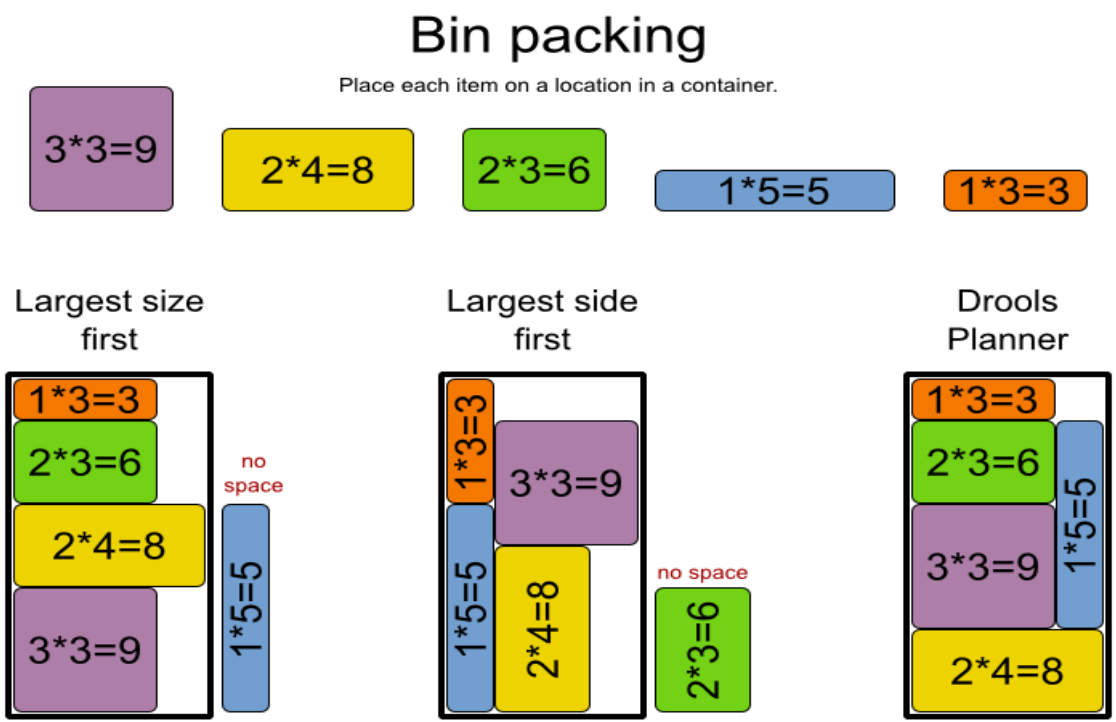
Swarm Manager

- Manages the Nodes in the cluster
- Uses Docker APIs to communicate with Docker Daemon on each node
- May be clustered

Scheduler

- Provides containers placing on the Nodes
- Pluggable architecture - Bring your own scheduler
- Each Scheduler contains of:
  - Placement strategy
  - Filter list

Scheduler – Bin Packaging



Scheduler - strategies

- **Bin Packaging**
  - Archive Nodes as compact as possible
- **Spread**
  - Distributes Nodes evenly
- **Random**
  - Only for debugging as a rule

Scheduler - Filters

- **Affinity Filter**
- **Constraint Filter**
- **Health Filter** – selects only healthy Nodes
- **Port Filter**

Constraint Filter

1. **Each Docker-host may have various list of tags**
   - OS, Storage (ssd, disk), kernel, env
2. **You may choose host by selecting criteria of tags**
   - storage=ssd
   - region=us-east
   - environment=production

- **Standard tags**:
  - node ID or node Name (using key “node”)
  - storagedriver
  - executiondriver
  - kernelversion
  - operatingsystem

Affinity filters

1. **Affinity** and **anti-affinity** rules
   - Locate db with nginx
   - Locate db with selected image
   - Don’t locate container if label==frontend
2. **Restrictions may be strict and soft**

Port filters

1. Place the container only if certain public port is free on the Node . For instance 80 or 443

Dependency if filters

1. You can apply dependency on the existing containers
2. Shared volumes
    1. Links
    2. Shared network stacks

Resource Management

- **RAM**
  - docker run -m 1g
- **CPU**
  - docker run -c 1
- **Ports**
  - docker run -p 80:80

Service Discovery

- Token Based
- etcd based
- Zookeeper based
- Consul Based
- File Based
- Bring your own?

High Availability

- **Multiple Swarm Managers**
  - Similar to Master-Master replication
  - Chooses new Master If Master Manager goes offline
  - **Works only with**
    - Consul
    - Etcd
    - Zookeeper
- **For the correct selection of a new Master consensus is required**

Requests Routing

- There are no standard patterns
- DIY

How to deploy?

- Manually with Docker
- Docker machine
- OpenStack Magnum

Swarm Advantages

1. Allows you to describe your application lifecycle in details
2. Easy to manage extensibility in affinity & anti-affinity part instead of Kubernetes
3. Default Docker orchestrator

Swarm limitations

1. Complicated manual cluster deployment
2. Complicated automatic horizontal scaling set up
3. All the actions have to be performed in CLI
4. There is no functionality that allows application deployment declarative description.
5. No support of health-checks
6. No automatic rescheduling of inactive Nodes
7. Relatively fresh release with some issues