Software Maintenance in the Serverless Era: DevOps, Infrastructure Code, and More

Damian A. Tamburri
Technical University Eindhoven and
Jeronimus Academy of Data Science (NL)

Agenda for Today!

Jheronimus Academy of Data Science

- Some context…
- DevOps Explained
- DevOps Tactics in Action: Maintaining Infrastructure-as-Code & TOSCA
- DevOps Tactics in Action: Maintaining Microservices & Serverless
- Conclusions & Take-home messages

Hot Topic for Today!

"DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality."

L. Bass et Al. [11]

Hot Topic for Today!

Jheronimus Academy of Data Science

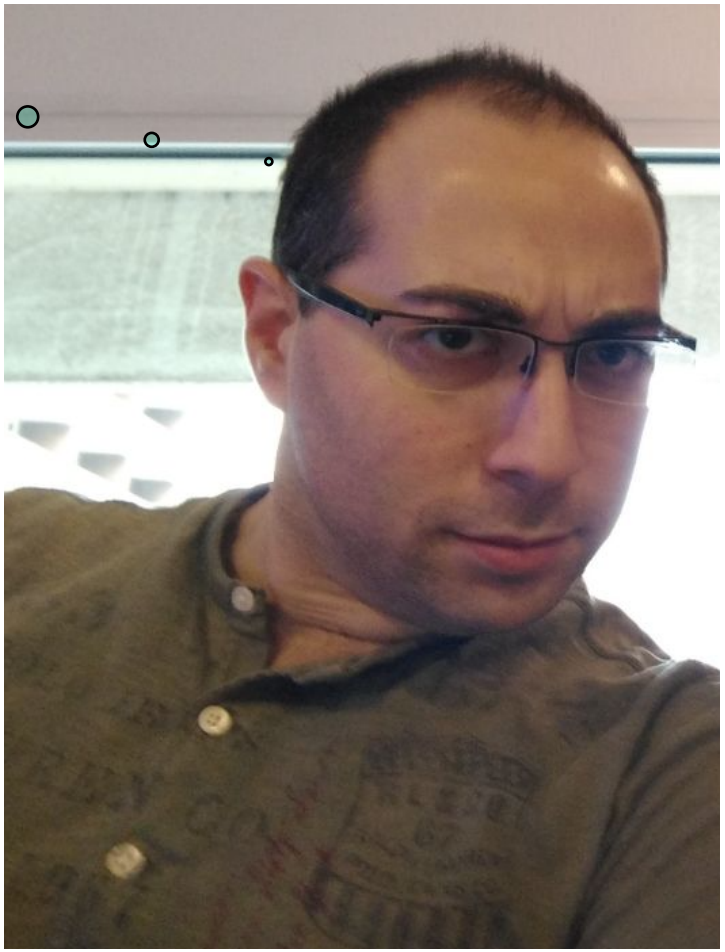
- *"Say What???????"*

My Face when I heard of it...

Jheronimus Academy of Data Science

- DevOps?

"Say What???????"



But first... A bit of history!

- Let’s take a step back to where it all began...

It's 2013...

Jheronimus Academy of Data Science

- And...

It's 2013...

- And...

It's 2013…

- And…

EU election 2014: Italy's Renzi triumphs as comic Grillo loses ground

New PM scores sweeping victory in election, leaving former comic Beppe Grillo's anti-establishment 5-Star Movement and Silvio Berlusconi's Forza Italia trailing

Meanwhile in Software Engineering... Top failure causes*

Jheronimus Academy of Data Science

- Unrealistic deadlines, e.g., imposed by someone external to the technical staff
- Requirements & people change (too) often
- Effort and resources have been estimated in an overly optimistic way,
- Risks have not been taken into account from the start of the project.
  - Risks can be technical or human difficulties
- Communication problems among staff members
- Difficulty by the management to recognize recurrent delays and take immediate action
- Subversive stakeholders

*Gartner Report 2013

Meanwhile in Software Engineering...
Top failure causes* - An Example!

Jheronimus Academy of Data Science

- **Unrealistic deadlines, e.g., imposed by someone external to the technical staff**
- **Requirements & people change (too) often**
- Effort and resources have been estimated in an overly optimistic way,
- Risks have not been taken into account from the start of the project.
  - Risks can be technical or human difficulties
- **Communication problems among staff members**
- Difficulty by the management to recognize recurrent delays and take immediate action
- **Subversive stakeholders**

*Gartner Report 2013

Meanwhile in Software Engineering...
Top failure causes* - An Example!

Jheronimus Academy of Data Science

(UNFORESEEN) OVERHEAD COST: 174,000,000 $ (give or take)*

*http://www.cio.com/article/2380827/developer/6-software-development-lessons-from-healthcare-gov-s-failed-launch.html

How is DevOps a response?

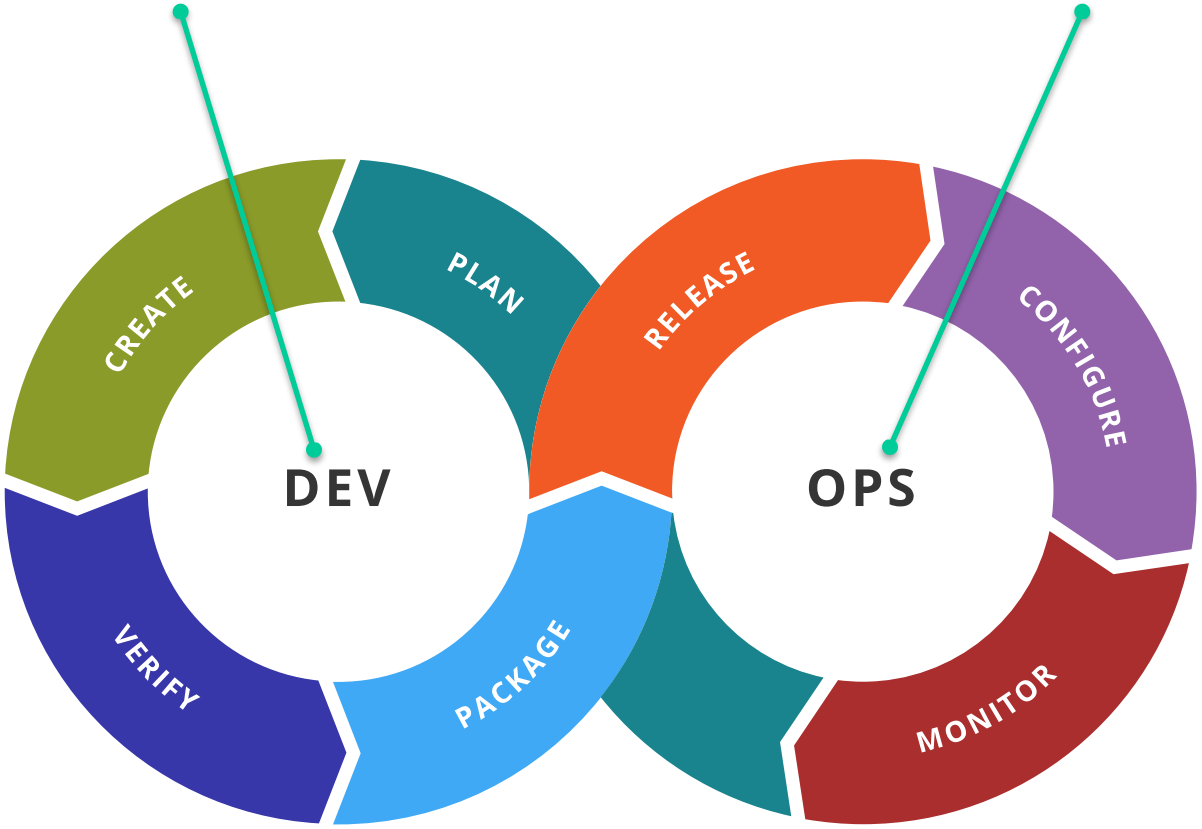
"DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality."

L. Bass et Al. [11]

What is DevOps?

Dev Goal: "Prepare a Software Architecture designed to be immediately deployable"

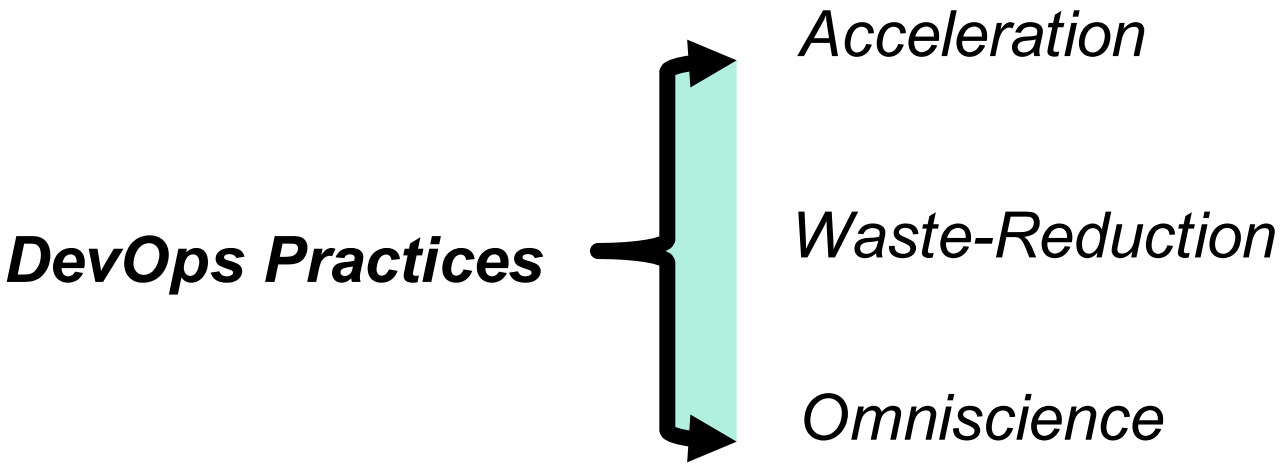
Ops Goal: "Observe the architecture runtime and provide Ops feedback to Dev…*then improve architecture continuously*"



What is DevOps?

*"DevOps is a set of **practices** intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality."*

L. Bass et Al. [11]



DevOps Practices: Let's take a look

Jheronimus Academy of Data Science

- **Acceleration Tactics**
  - Use Faster Organization: Merge Dev+Ops Teams…
  - Infrastructure-as-Code
  - Use Continuous Integration Tools
  - Use Continuous Deployment Tools
  - Use Continuous Testing Tools
  - …
- **Waste Reduction Tactics**
  - Canary Testing
  - A/B Testing
  - Reduce Documentation
  - Minimalistic Architecting ➔ Microservices
  - …

DevOps Practices: Let's take a look

Jheronimus Academy of Data Science

- **Omniscience Tactics**
  - Monitor Everything (what we call *Omniscient DevOps Analytics*)
  - Monitoring-as-a-service
  - On-The-Fly Risk Engineering
- ...

DevOps Practices: Let's throw in another one…

- **Omniscience Tactics**
  - Monitor Everything
  - Monitoring-as-a-service
  - On-The-Fly Risk Engineering
  - **Continuous Architecting**
- …

Maintenance Challenges

- More granular systems continuously deployed:
  - Maintain the systems

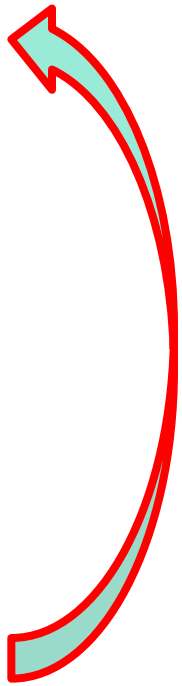
This is a classic… accounting typically for ~70% tot. Cost for software

Maintenance Challenges

- More granular systems continuously deployed:
  - Maintain the systems
  - Maintain the pipelines around the systems
  - Concurrency between the above two
- Larger and more diverse organisations:
  - «Maintain» the people
  - Measure and Track the organisational structure

Maintenance Challenges

- More granular systems continuously deployed:
  - Maintain the systems
  - Maintain the pipelines around the systems
  - Concurrency between the above two
- Larger and more diverse organisations:
  - «Maintain» the people
  - Measure and Track the organisational structure



- **Socio-Technical Alignment!**

DevOps Processes & Practices: Digging Further

- Continuous Architecting

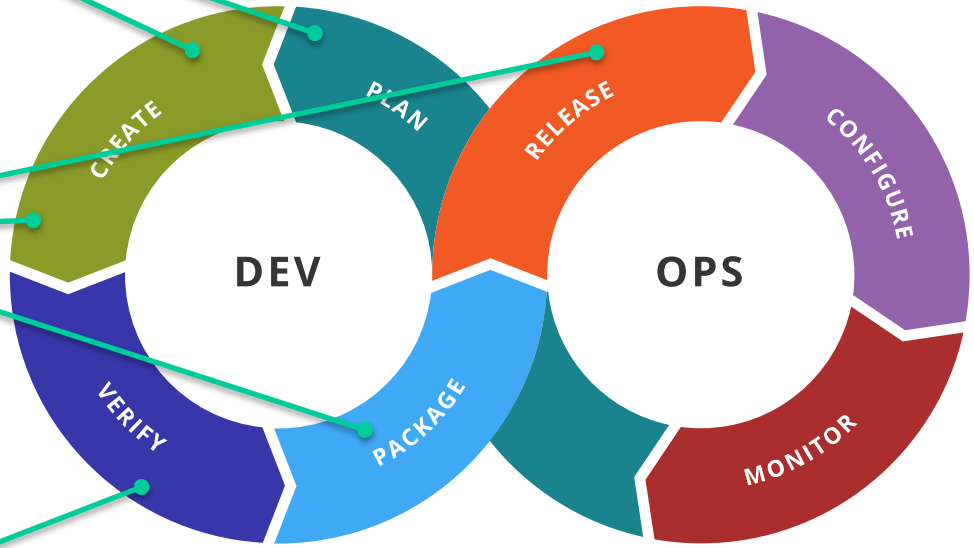
Def. *"architect for test, build and deploy, take quality attributes into account, take advantage of feedback from runtime" [1]*

- Continuous Integration

Def. *"merge* all developer work-copies to a shared mainline frequently*" [4]*
***Examples. Apache Jenkins**, **Hudson,** etc.*

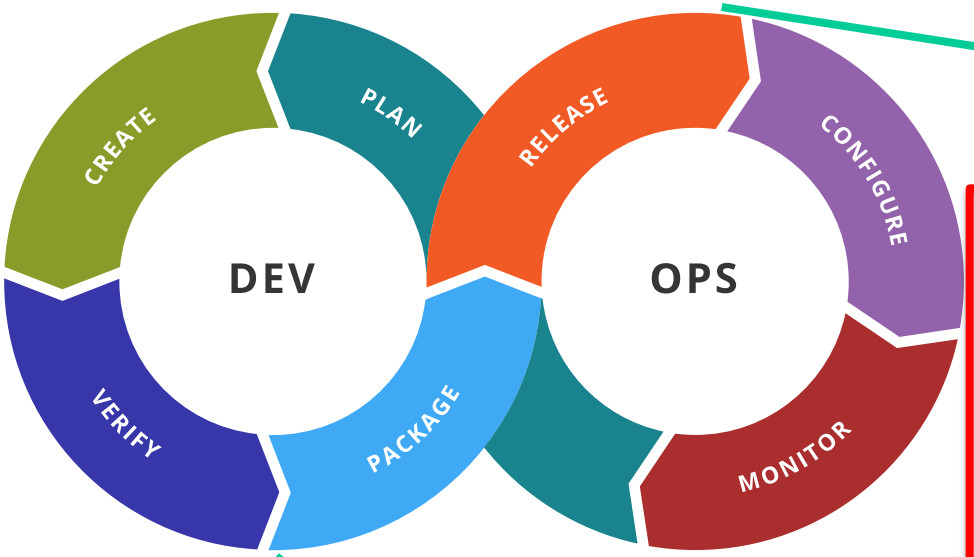
- Continuous Testing

Def. *"run tests as part the build pipeline so that every check-in and deployment is validated" [3]*
***Examples. Selenium+GitHub+LI-API**, etc.*



...

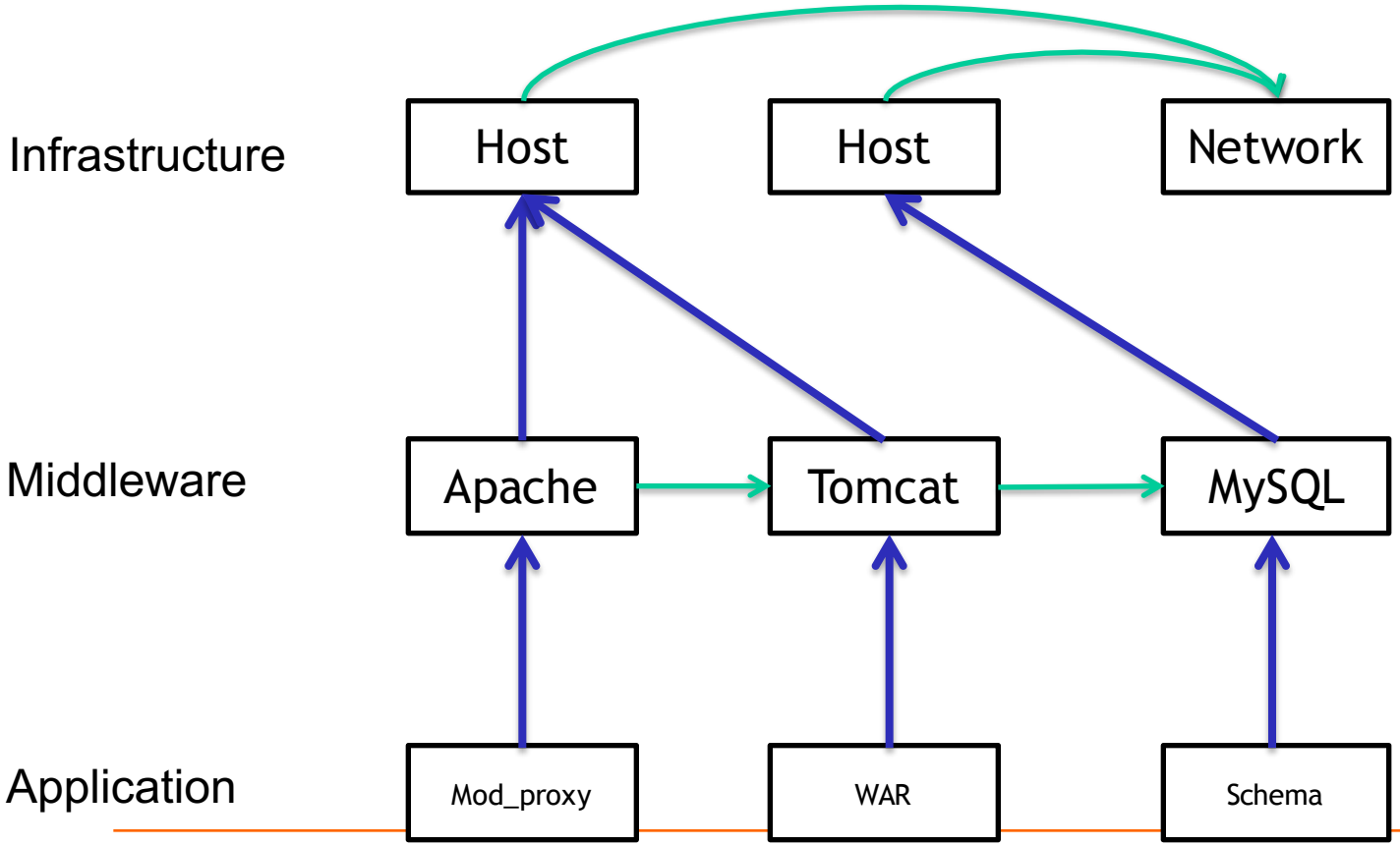
DevOps process and toolchain



Key DevOps Tactic: Infrastructure Code

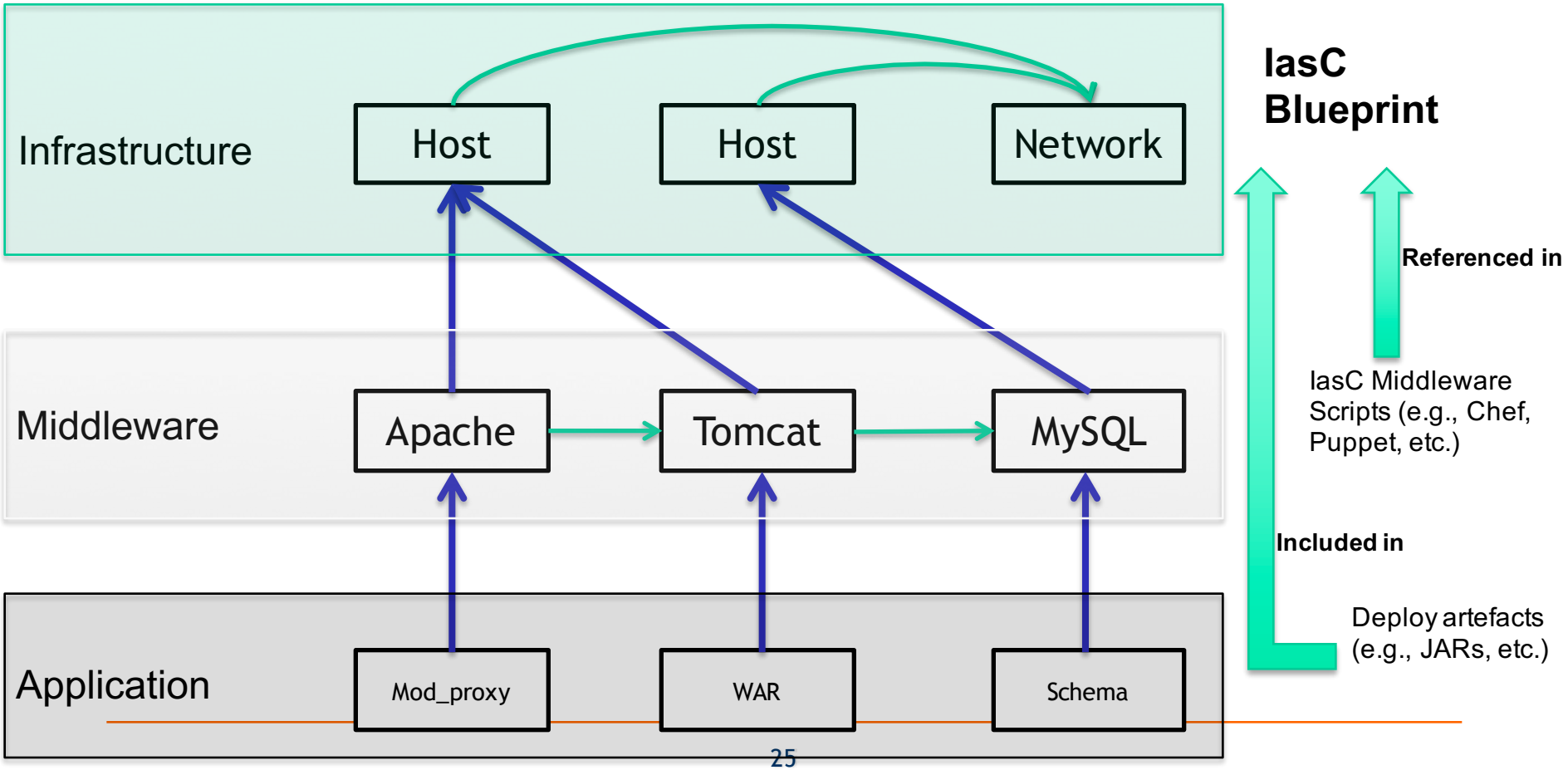
Jheronimus Academy of Data Science

➔ **An Application Deployment Topology**, i.e., "a graph of physical artefacts that need support for several lifecycle phases (e.g., procurement, installation, configuration, deployment, undeployment, teardown, etc.)" [6]



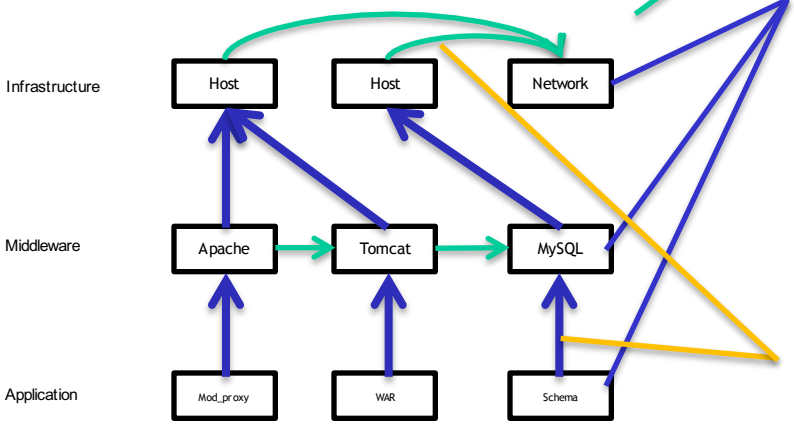
Towards standard Infrastructure Code

➔ **Infrastructure-as-code**, i.e., "a blueprint detailing physical artefacts, all scripts for all lifecycle phases and all artefacts needed for deployment" [6]



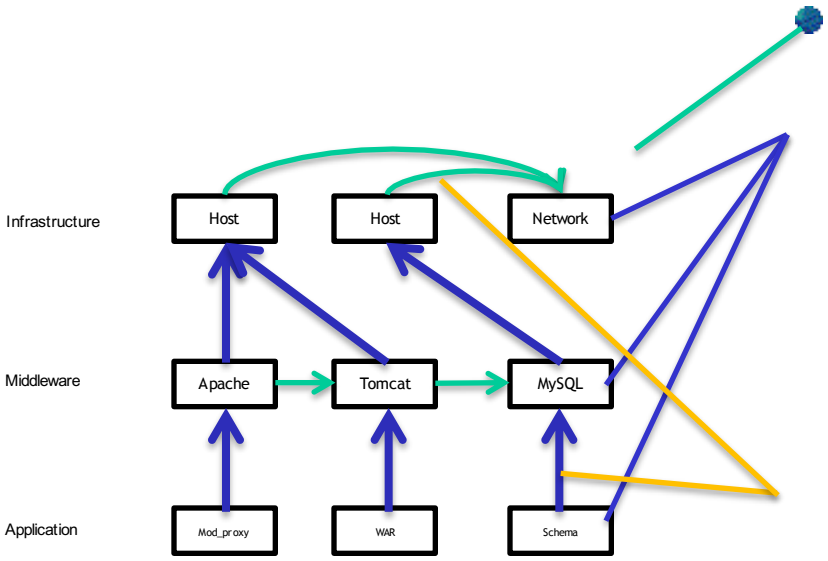
Where Does IaC fit into?

Here's What We've Seen there…



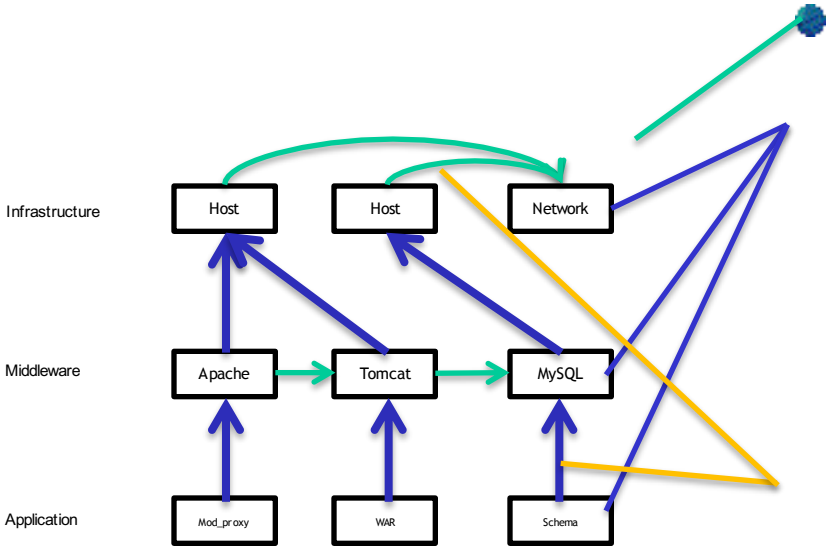
- An application topology
- 3 layers
  - Infrastructure (Cloud or DC objects)
  - Platform or Middleware (App containers)
  - Application modules, schemas and configurations
- Relationships between components:
  - What's hosted on what or installed on what
  - What's connected to what

Where Does IaC fit into?



- ***Many*** available formats (70+)*:
  - Saltstack
  - Kubernetes
  - CloudFormation
  - Cloudify
  - Ubicity
  - xOpera
  - HEAT
  - …

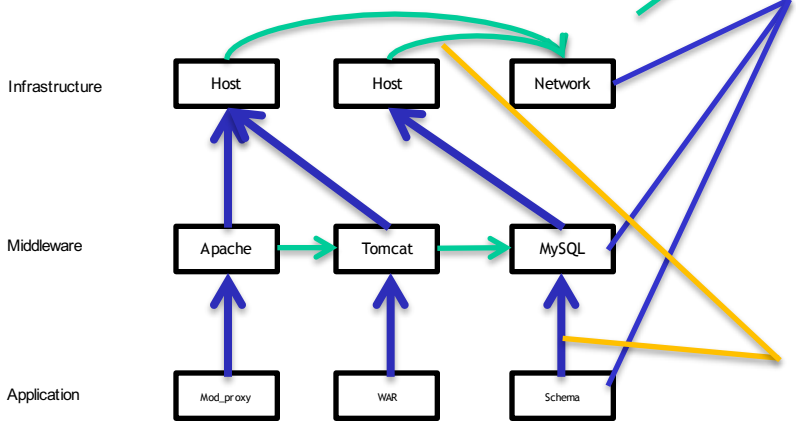
From IaC to TOSCA



TOSCA: "**T**opology and **O**rchestration **S**pecification for **C**loud **A**pplications"

- **Only ONE Standard!**

From IaC to TOSCA (cont'd)



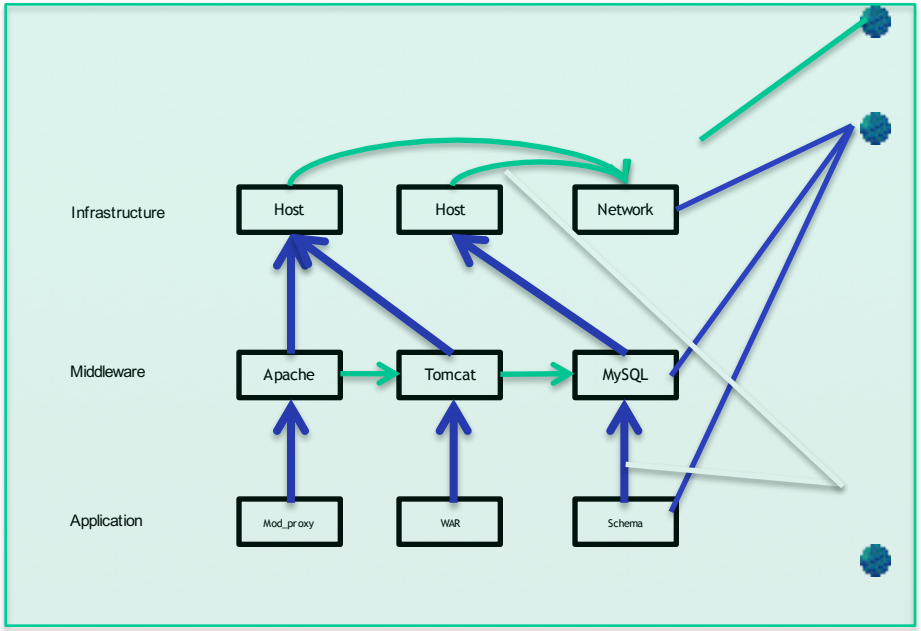
- An application topology
- 3 layers
  - Infrastructure (Cloud or DC objects)
  - Platform or Middleware (App containers)
  - Application modules, schemas and configurations
- Relationships between components:
  - What's hosted on what or installed on what
  - What's connected to what

TOSCA: "**T**opology and **O**rchestration **S**pecification for **C**loud **A**pplications"

Where Does TOSCA fit into?

Jheronimus Academy of Data Science

A single abstract language for IaC



TOSCA: "**Topology** and **O**rchestration **S**pecification for **C**loud **A**pplications"

An application topology

3 layers

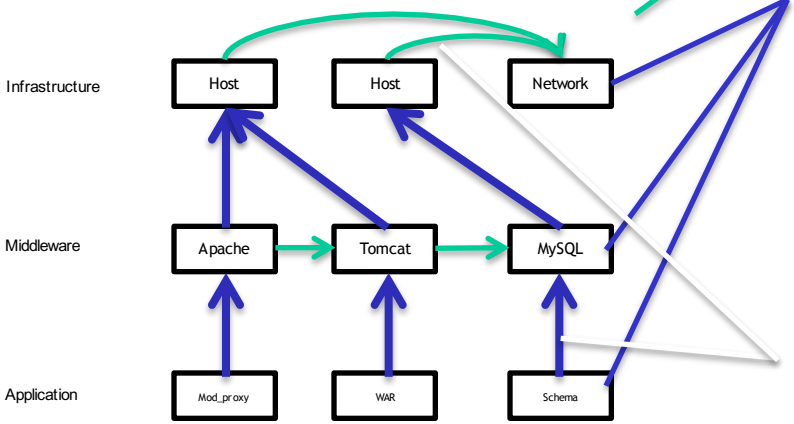
- Infrastructure (Cloud or DC objects)
- Platform or Middleware (App containers)
- Application modules, schemas and configurations

Relationships between components:

- What's hosted on what or installed on what
- What's connected to what

Where Does TOSCA fit into?

A single abstract language for IaC

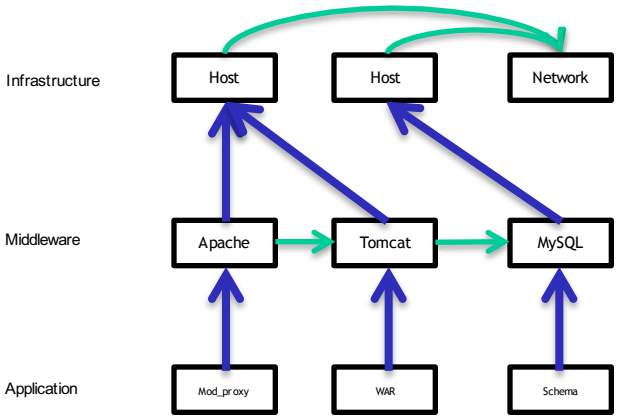


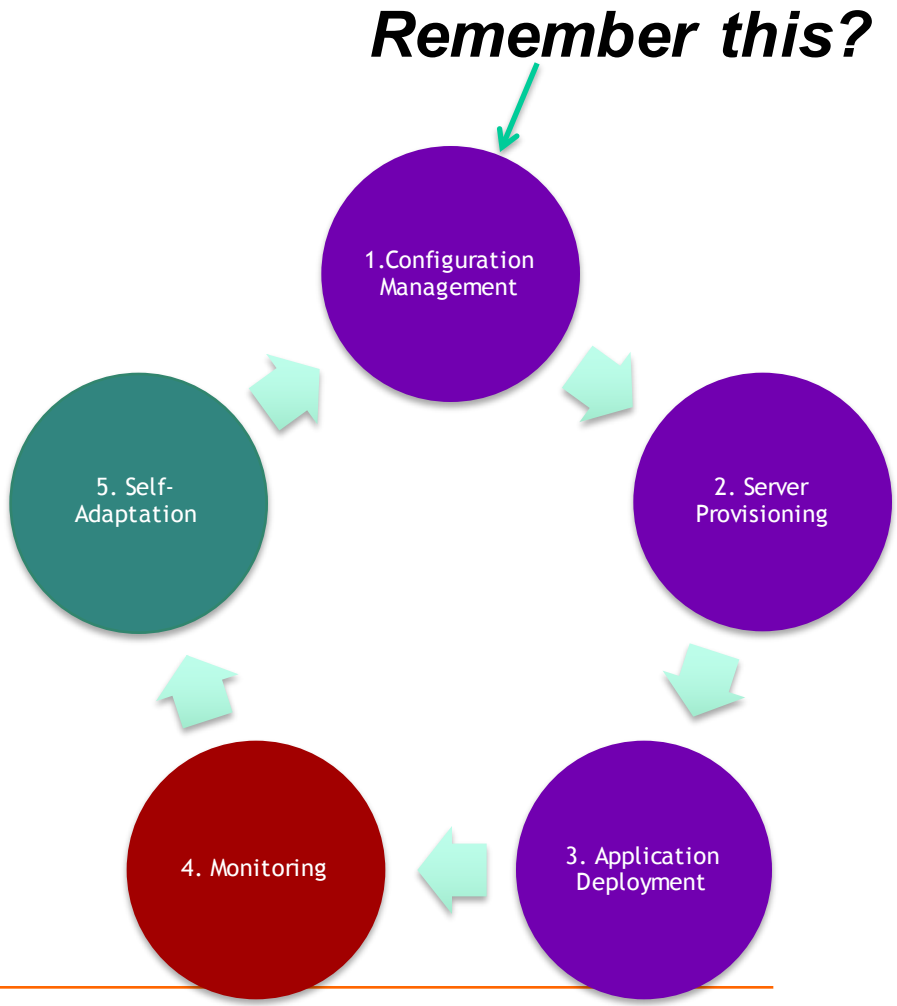
- An application topology
- 3 layers
  - Infrastructure (Cloud or DC objects)
  - Platform or Middleware (App containers)
  - Application modules, schemas and configurations
- Relationships between components:
  - What's hosted on what or installed on what
  - What's connected to what

TOSCA: "**T**opology and **Orchestration** **S**pecification for **C**loud **A**pplications"

Where Does TOSCA fit into?

A single abstract language for IaC



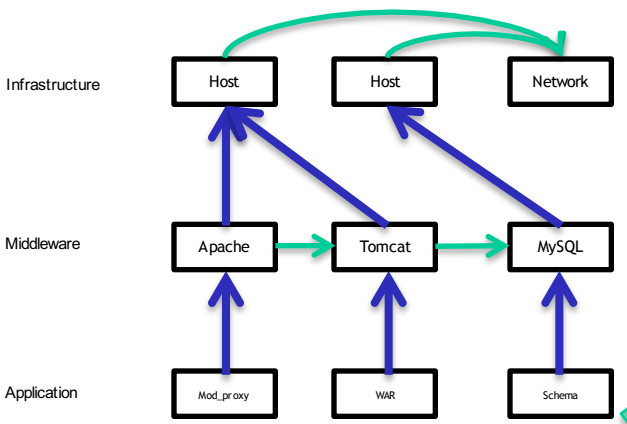


TOSCA: "**T**opology and **Orchestration** **S**pecification for **C**loud **A**pplications"

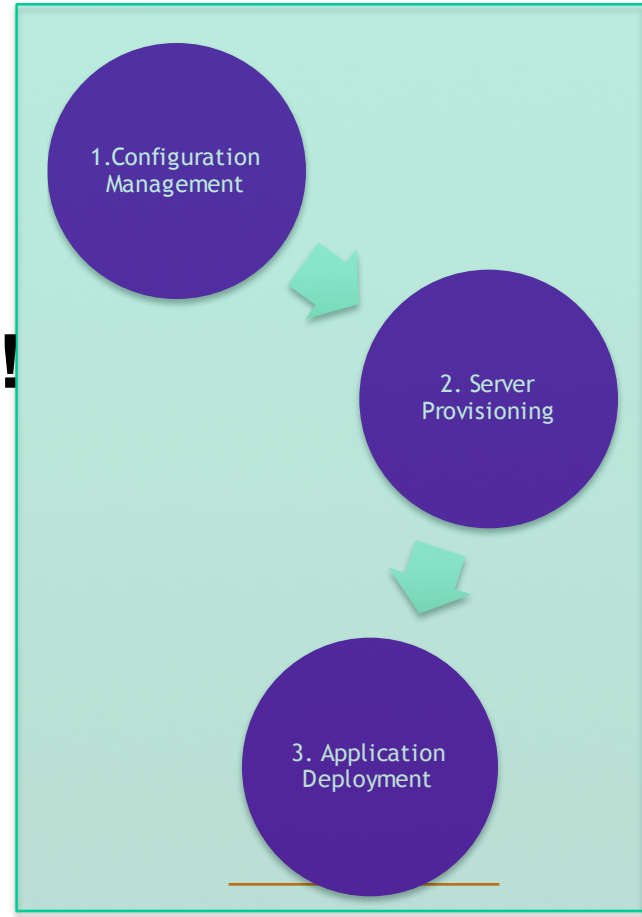
Where Does IaC and TOSCA fit into?

Jheronimus Academy of Data Science

A single abstract language for IaC



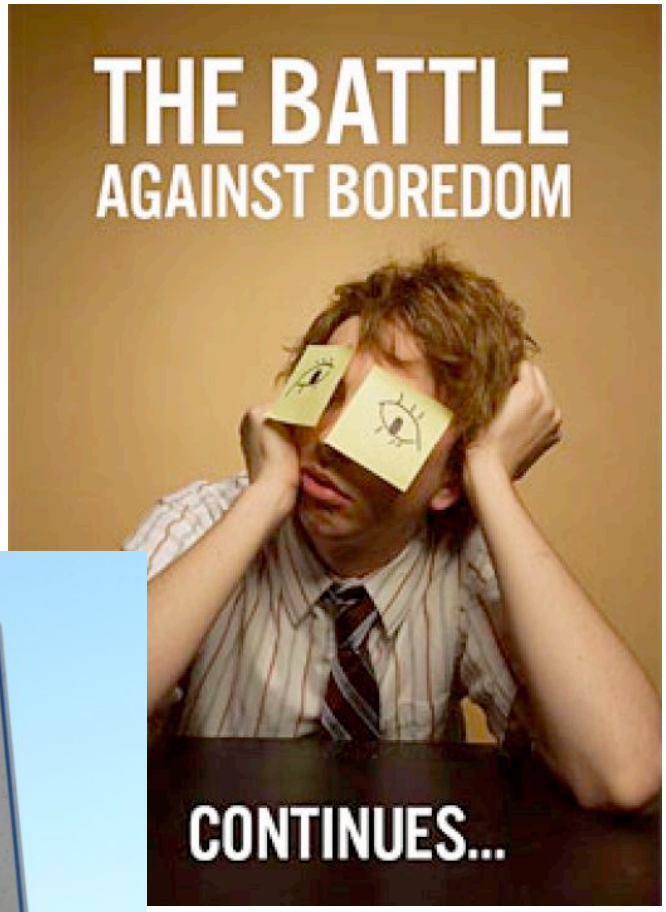
Orchestration!



TOSCA: "**T**opology and **O**rchestration **S**pecification for **C**loud **A**pplications"

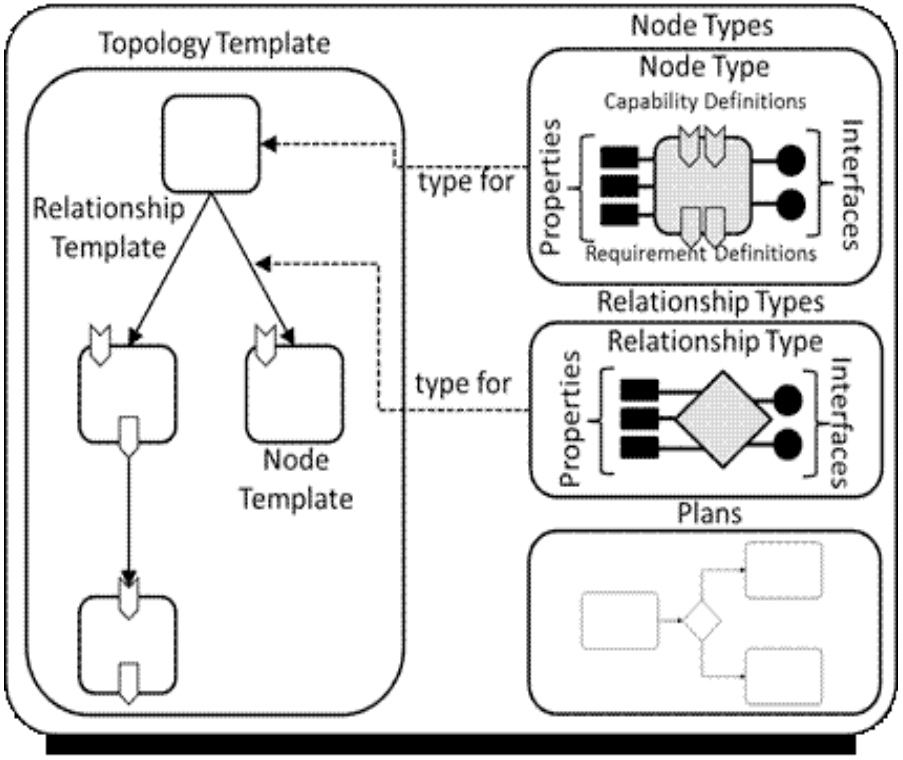
Bored yet?

- 10 min Break?



Coming up next... TOSCA, IaC Explained!

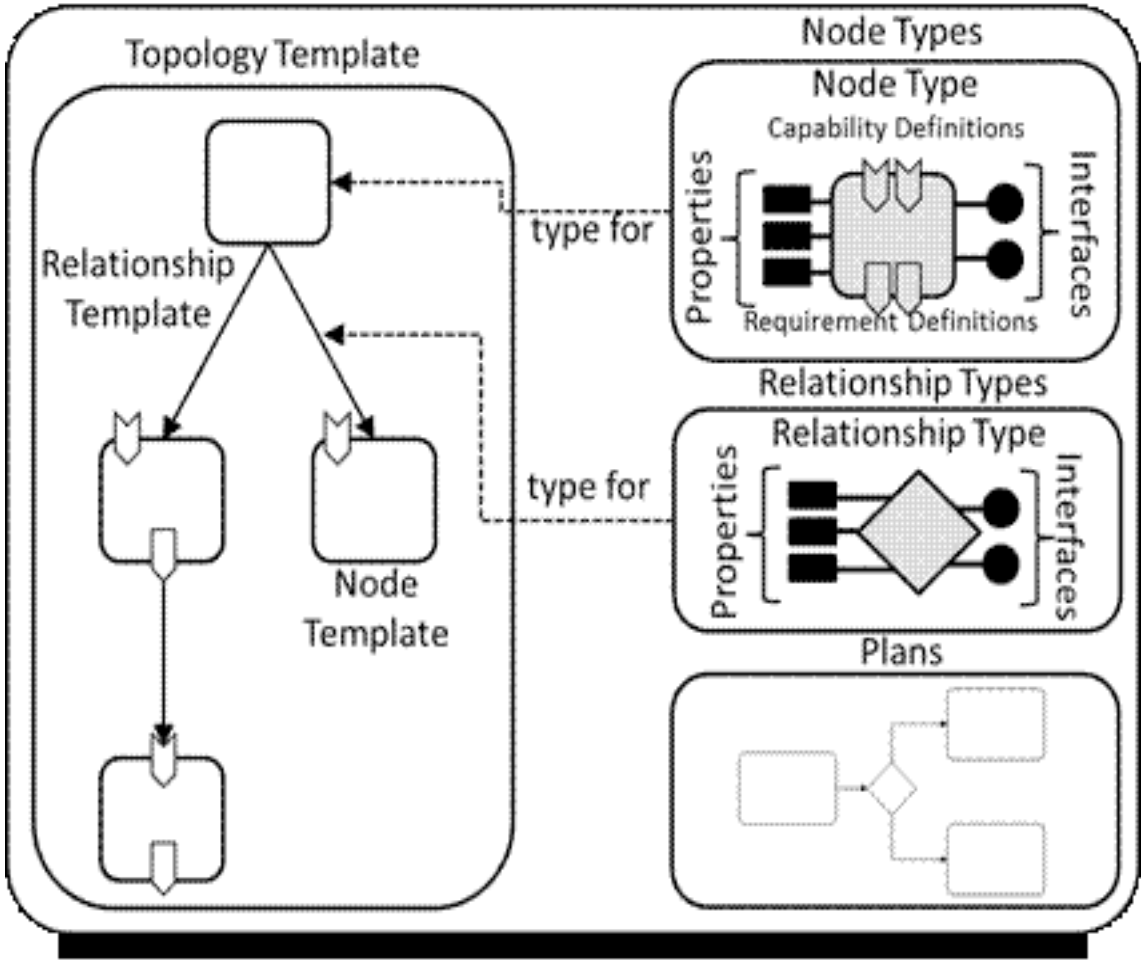
What's in a TOSCA Topology?



- component in the topology are called Nodes
- Each Node has a Type (e.g. Host, BD, Web server).
  - The Type is abstract and hence portable
  - The Type defines Properties and Interfaces
- An Interface is a set of hooks (named Operations)
- Nodes are connected to one another using Relationships

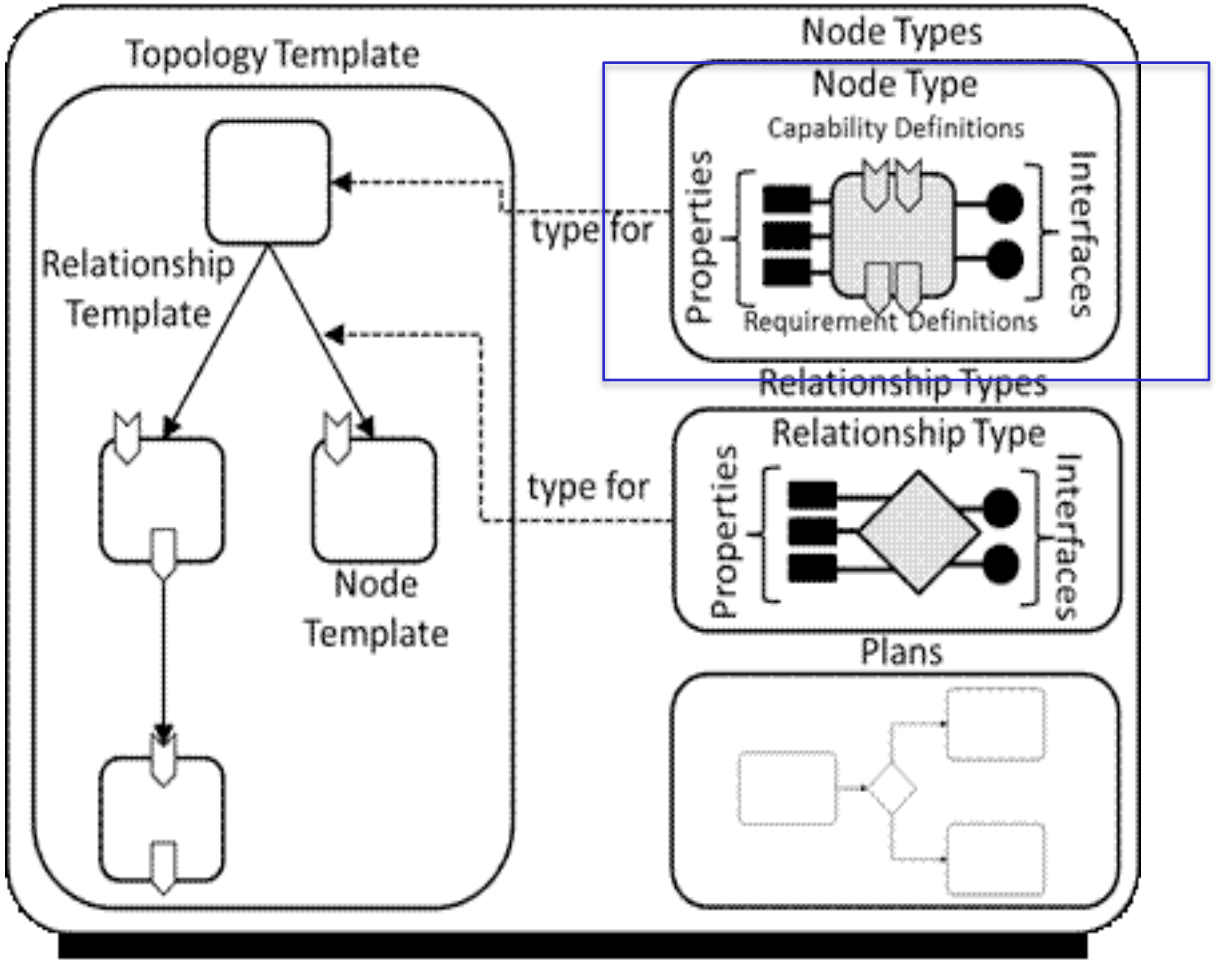
TOSCA, IasC Standard

TOSCA Service Template [7]



TOSCA Core Ingredients

TOSCA Service Template



Node Type

- Describes a Cloud or Software type (e.g. Server or Apache)
- Maps the type to the actual impl. of the lifecycle interface

```
tosca.interfaces.node.Lifecycle:
  create:
    description: Basic lifecycle create operation.
  configure:
    description: Basic lifecycle configure operation.
  start:
    description: Basic lifecycle start operation.
  stop:
    description: Basic lifecycle stop operation.
  delete:
    description: Basic lifecycle delete operation.
```

Node Type (cont.)

- Defines properties as YAML maps
- Might define capabilities (What it can provide to other nodes)

```
tosca.nodes.DBMS
  derived_from: tosca.nodes.SoftwareComponent
  properties:
    dbms_root_password:
      type: string
      description: the root password for the DBMS service
    dbms_port:
      type: integer
      description: the port the DBMS service will listen to for data and requests
  capabilities:
    host:
      type: Container
      containee_types: [ tosca.nodes.Database ]
```

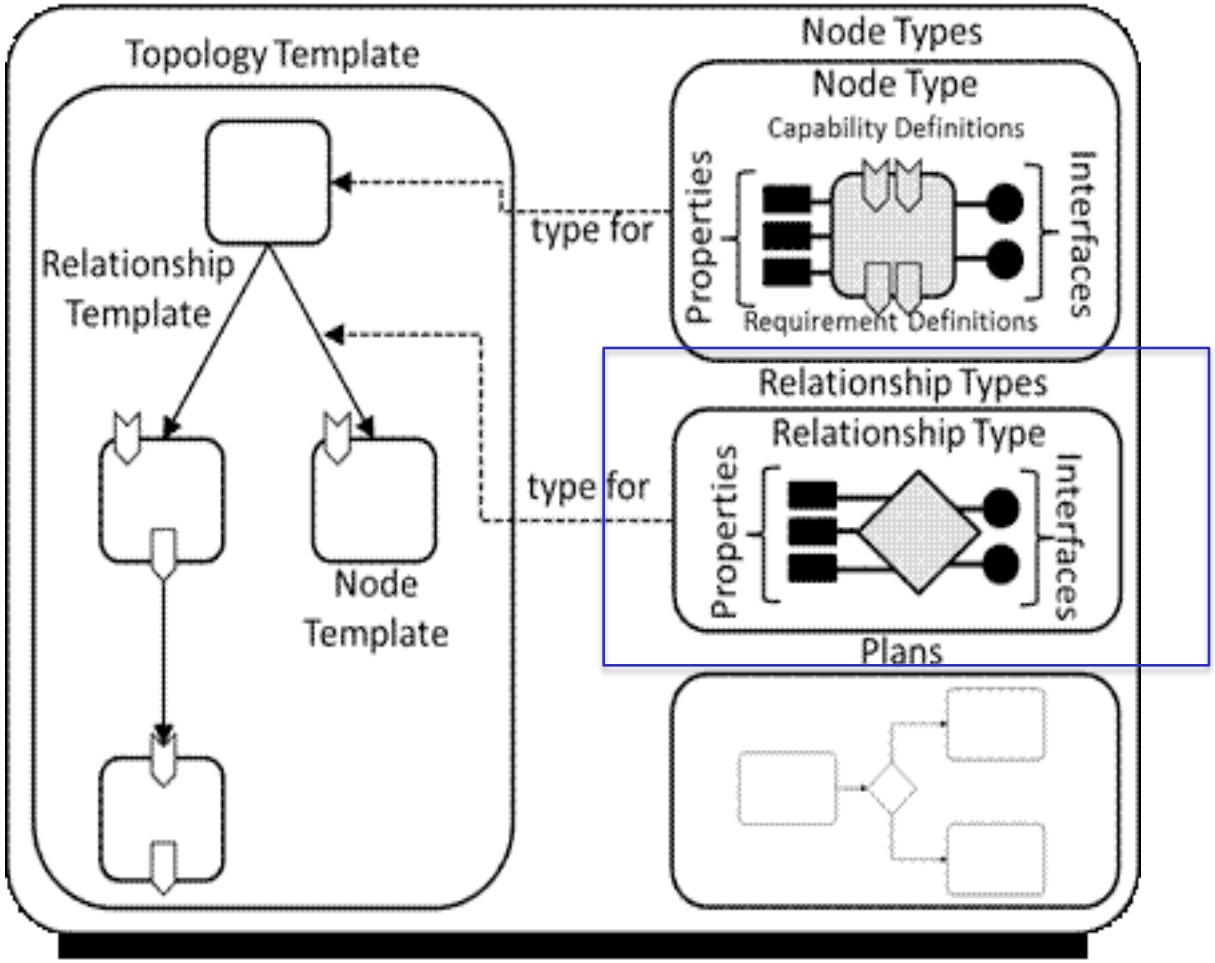
Node Type (cont.)

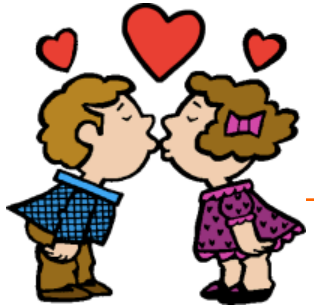
- Might define requirements (what it needs from other nodes)

```
tosca.nodes.Database:
  derived_from: tosca.nodes.Root
  properties:
    db_user:
      type: string
      description: user account name for DB administration
    db_password:
      type: string
      description: the password for the DB user account
    db_port:
      type: integer
      description: the port the underlying database service will listen to data
    db_name:
      type: string
      description: the logical name of the database
  requirements:
    - host: tosca.nodes.DBMS
  capabilities:
    - database_endpoint: tosca.capabilities.DatabaseEndpoint
```

TOSCA Core Ingredients

TOSCA Service Template





Relationship Type

Jheronimus Academy of Data Science

- Requirements and Capabilities are an implicit way to describe relationships
- Usually you need the explicit way
  - You need hooks to configure the source or target node or both
- So relationships have types and interfaces as well

Relationships (cont.)

Jheronimus Academy of Data Science

- The basic relationship types are:
  - **dependsOn** – abstract type and its sub types:
  - **hostedOn** – a node is contained within another
  - **connectsTo** – a node has a connection configured to another
- The basic interface is configure
  - **preconfigure_source**, **preconfigure_target**
  - **postconfigure_source**, **postconfigure_target**
  - **add_target**, **remove_target**

Node Templates

Jheronimus Academy of Data Science

- An instance of a type (like Object to Class)
- Has specific properties
- Has artifacts:
  - What to install
  - How to install (mapped to interface hooks)
- Has requirements and capabilities (or relationships)

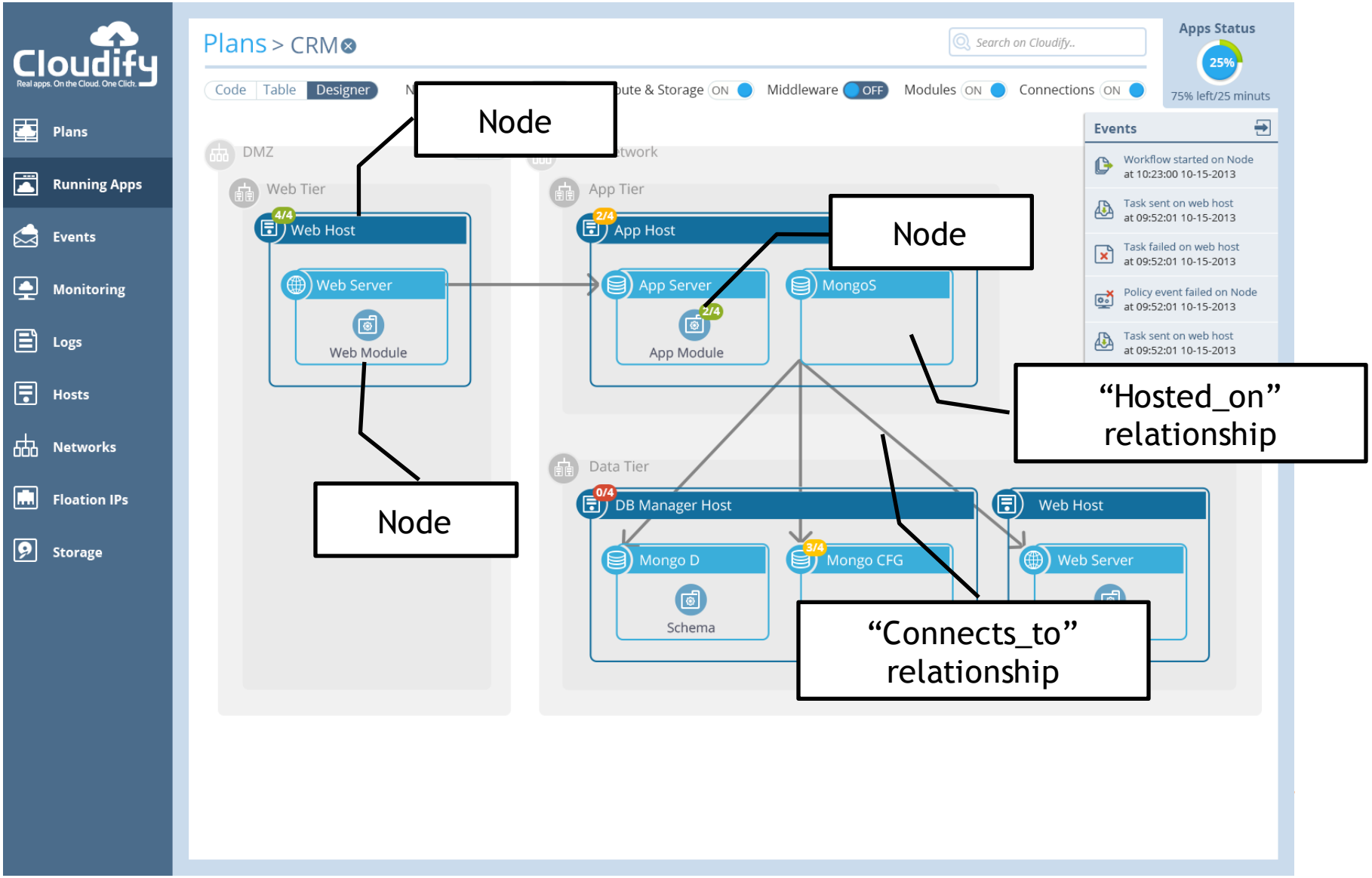
Node Template (Examples)

Jheronimus Academy of Data Science

```
node_templates:
  wordpress:
    type: tosca.nodes.WebApplication.WordPress
    properties:
      # omitted here for sake of brevity
    requirements:
      - host: apache
      - database: wordpress_db
        interfaces:
          tosca.interfaces.relationships.Configure:
            pre_configure_source: scripts/wp_db_configure.sh
```

```
node_templates:
  wordpress:
    type: tosca.nodes.WebApplication.WordPress
    properties:
      # omitted here for sake of brevity
    requirements:
      - host: apache
      - database: wordpress_db
        relationship_type: my.types.WordpressDbConnection
```

Translated to TOSCA



Workflows

- Imperative flow algorithm
- Using a workflow engine
- Timing the invocation of operations on different node
- **Examples?** Any BPMN specification!

- *But… Considered out of scope for the standard (but currently debated, two factions formed in the TOSCA TC)*

Policies

Jheronimus Academy of Data Science

- Brings monitoring to the orchestration as input
- Ongoing evaluation of Rules
- Enforce SLA, Health, and anything else
- Can invoke more processes
- **Standard Structure:** <Event><Condition><Action>
- **Standard Types:**
  - Access-Control;
  - Placement;
  - QoS (Quality) or (Continuity) CoS;
- **Example?**

TOSCA Policy Example

Jheronimus Academy of Data Science

Event Type

```
<event_type_name>:
  derived_from: <parent_event_type>
  version: <version_number>
  description: <policy_description>
```

Policy Definition

```
<policy_name>:
  type: <policy_type_name>
  description: <policy_description>
  properties:      <property_definitions>
  # allowed targets for policy association
  targets: [ <list_of_valid_target_templates> ]  *
  triggers:
    <trigger_symbolic_name_1>:
        event: <event_type_name>
        # TODO: Allow a TOSCA node filter here
        # required node (resource) to monitor
        target_filter:
          node: <node_template_name> <node_type>
          # Used to reference another node related to
          # the node above via a relationship
          requirement: <requirement_name>
          # optional capability within node to monitor
          capability: <capability_name>
        # required clause that compares an attribute
        # with the identified node or capability
        # for some condition
        condition: <constraint_clause>
        action:
          # a) Define new TOSCA normative strategies
          # per-policy type and use here OR
          # b) allow domain-specific names
          <operation_name>: # (no lifecycle)
              # TBD: Do we care about validation of types?
              # If so, we should use a TOSCA Lifecycle type
              description: <optional description>
              inputs: <list of property assignments >
              implementation: <script> | <service_name>
    <trigger_symbolic_name_2>:
    ...
    <trigger_symbolic_name_n>:
```

Event
name of a normative TOSCA Event Type

Condition
described as a constraint of an attribute of the node (or capability) identified by the filter.

Action
Describes either:
a)a well-known strategy
b)an implementation artifact (e.g., scripts, service) to invoke

*with optional property definitions as **inputs** (to either choice)*

TOSCA Policy Example – Entities that compose Policy (Event, Condition, Action) model

Jheronimus Academy of Data Science

Event Type

```
<event_type_name>:
  derived_from: <parent_event_type>
  version: <version_number>
  description: <policy_description>
```

Policy Definition

```
<policy_name>:
  type: <policy_type_name>
  description: <policy_description>
  properties:      <property_definitions>
  # allowed targets for policy association
  targets: [ <list_of_valid_target_templates> ]  *
  triggers:
    <trigger_symbolic_name_1>:
        event: <event_type_name>
        # TODO: Allow a TOSCA node filter here
        # required node (resource) to monitor
        target_filter:
          node: <node_template_name> <node_type>
          # Used to reference another node related to
          # the node above via a relationship
          requirement: <requirement_name>
          # optional capability within node to monitor
          capability: <capability_name>
        # required clause that compares an attribute
        # with the identified node or capability
        # for some condition
        condition: <constraint_clause>
        action:
          # a) Define new TOSCA normative strategies
          # per-policy type and use here OR
          # b) allow domain-specific names
          <operation_name>: # (no lifecycle)
              # TBD: Do we care about validation of types?
              # If so, we should use a TOSCA Lifecycle type
              description: <optional description>
              inputs: <list of property assignments >
              implementation: <script> | <service_name>
    <trigger_symbolic_name_2>:
    ...
    <trigger_symbolic_name_n>:
```

Event
name of a normative TOSCA Event Type

Condition
described as a constraint of an attribute of the node (or capability) identified by the filter.

Action
Describes either:
a)a well-known strategy
b)an implementation artifact (e.g., scripts, service) to invoke

*with optional property definitions as **inputs** (to either choice)*

Example Senlin "scaling_out_policy_ceilometer.yaml"

using the Kubernetes "redis" example

Jheronimus Academy of Data Science

Target is a Kubernetes Pod of the tosca.groups.placement type

Symbolic name for the trigger *(could be used to reference an externalized version; however, this would violate a Policy's integrity as a "Security document"*

TOSCA Policy

```
my_scaling_policy:
  type: tosca.policies.scaling
  properties:  # normative TOSCA properties for scaling
    min_instances: 1
    max_instances: 10
    default_instances: 3
    increment: 1
  # target the policy at the "Pod"
  targets: [redis-master-pod]
  triggers:
    resize_compute:  # symbolic name
      event: tosca.events.resource.utilization
      target_filter:
        node: master-container
        requirement: host
        capability: Container
      condition: utilization greater_than 80%
      action:
        # map to SENLIN::ACTION::RESIZE
        scaleup: # logical operation name
          inputs:  # optional inputs parameters
            number: 1
            strategy: BEST_EFFORT
          Implementation: <script> | <service_name>
```

TOSCA normative event type (name) that would map to domain-specific names (e.g., OpenStack Ceilometer)

Find the attribute via the topology:

a) Navigate to node (directly or via the requirement name) and optionally the Capability name

b) The condition to map & register with the target monitoring service (e.g., Ceilometer)

Describe NODE to attach an alarm | alert | event to

i.e., Using the "node", "req", "cap" and "condition" keys would expressed as a descriptive "filter"

TODO:
Need a % data type for TOSCA

List optional input parms. here

Note: we combined the Senlin "Action" of ***SENLIN:ACTION:RESIZE*** *with the strategy:* ***BEST_EFFORT*** *to have one name*

more info online

Putting it All Together

Jheronimus Academy of Data Science

```
tosca_definitions_version: tosca_simple_yaml_1_0_0

description: >
  This TOSCA simple profile deploys nodejs, mongodb, elasticsearch, logstash and
kibana each on a separate server
  with monitoring enabled for nodejs server where a sample nodejs application is
running. The syslog and collectd are
  insatlled on a nodejs server.

imports:
  - tosca_base_type_definition.yaml
  - paypalpizzastore_nodejs_app.yaml
  - elasticsearch.yaml
  - logstash.yaml
  - kibana.yaml
  - collectd.yaml
  - rsyslog.yaml

dsl_definitions:
    host_capabilities: &host_capabilities
      # container properties (flavor)
      disk_size: 10 GB
      num_cpus: { get_input: my_cpus }
      mem_size: 4096 MB
    os_capabilities: &os_capabilities
      architecture: x86_64
      type: Linux
      distribution: Ubuntu
      version: 14.04

topology_template:
  inputs:
    my_cpus:
      type: integer
      description: Number of CPUs for the server.
      constraints:
        - valid_values: [ 1, 2, 4, 8 ]
  …
```

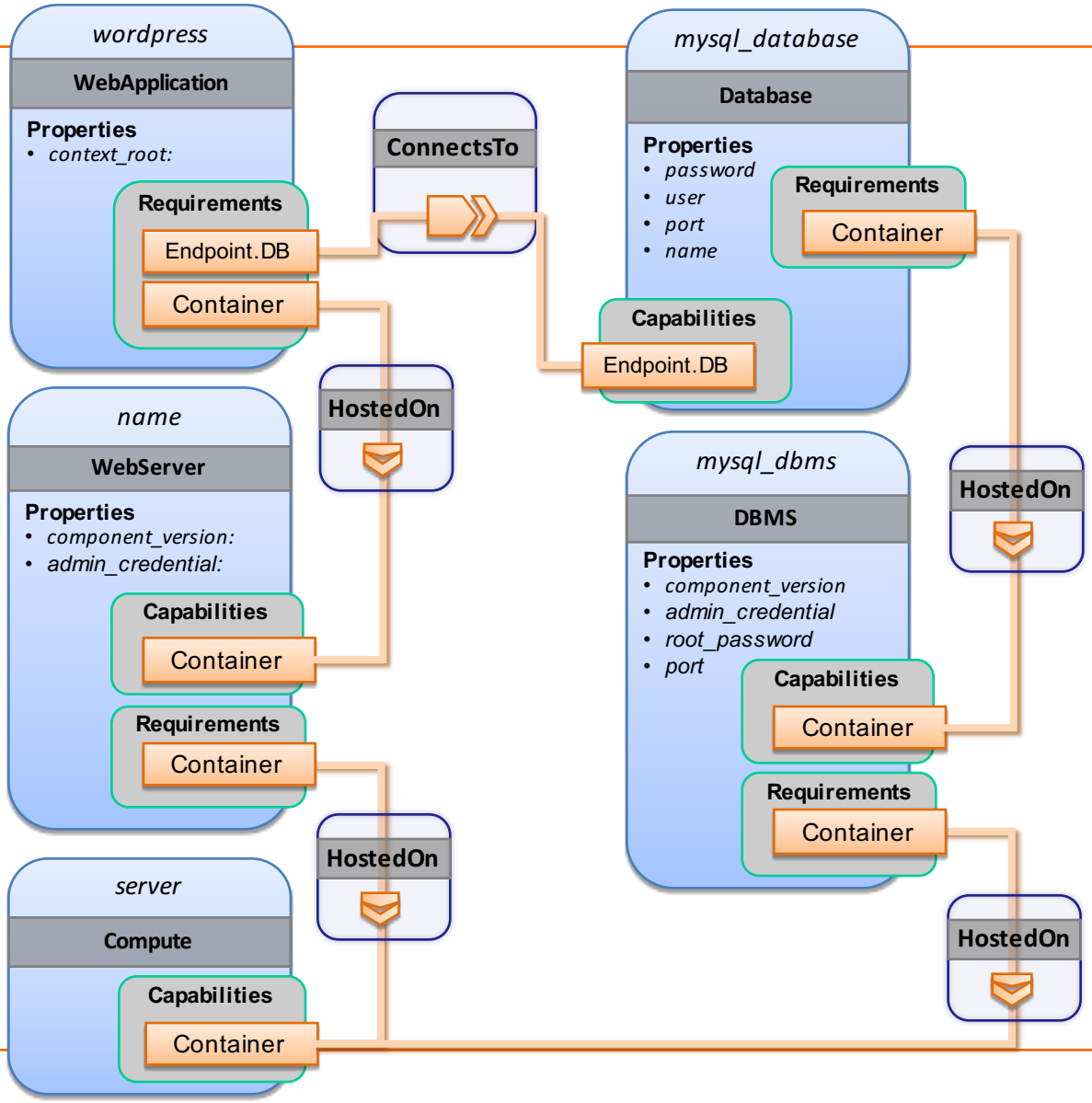
- TOSCA Template contains:
  - Application Topology
    - Nodes
      - Interfaces
      - Properties
      - Artifacts (Plugins in Cloudify)
    - Relationships
      - Interfaces
  - Workflows
  - Policies

TOSCA YAML IasC Examples

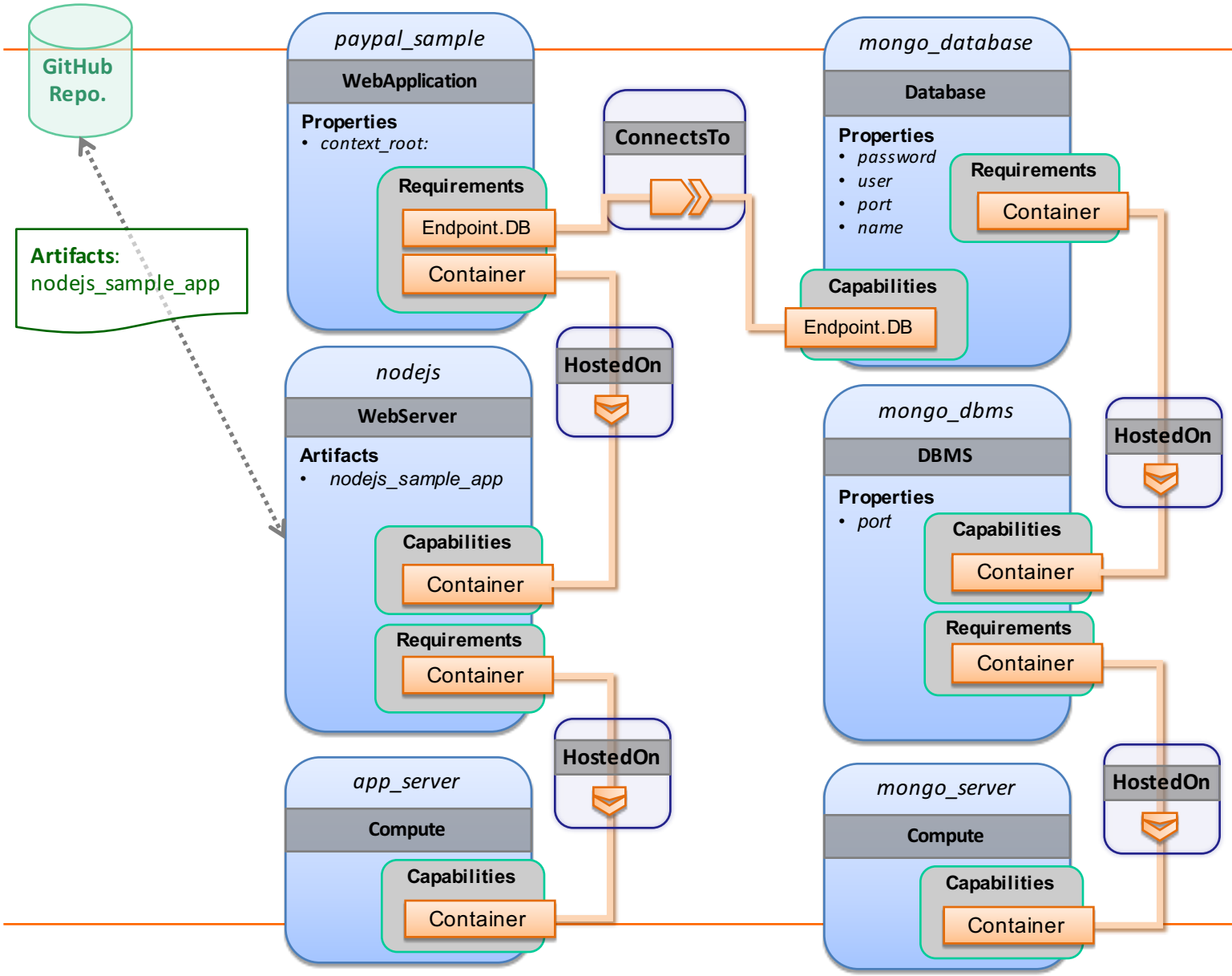
- WordPress+MySQL
- NodeJS App+MongoDB

WebServer-DBMS-1: WordPress - MySQL

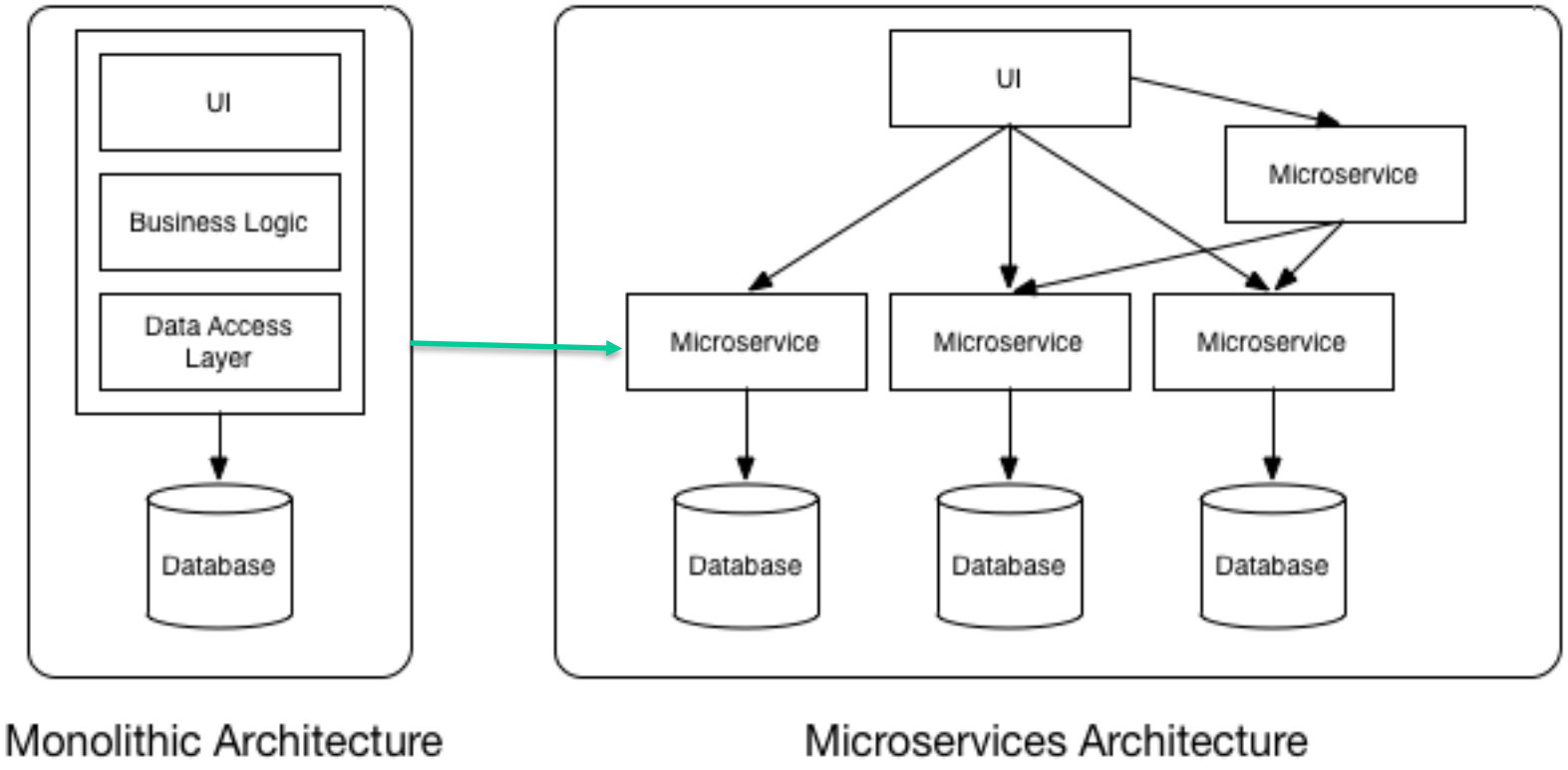
Jheronimus Academy of Data Science



WebServer-DBMS-3:
Nodejs - MongoDB



IaC in action: Continuous Architecting & Microservices

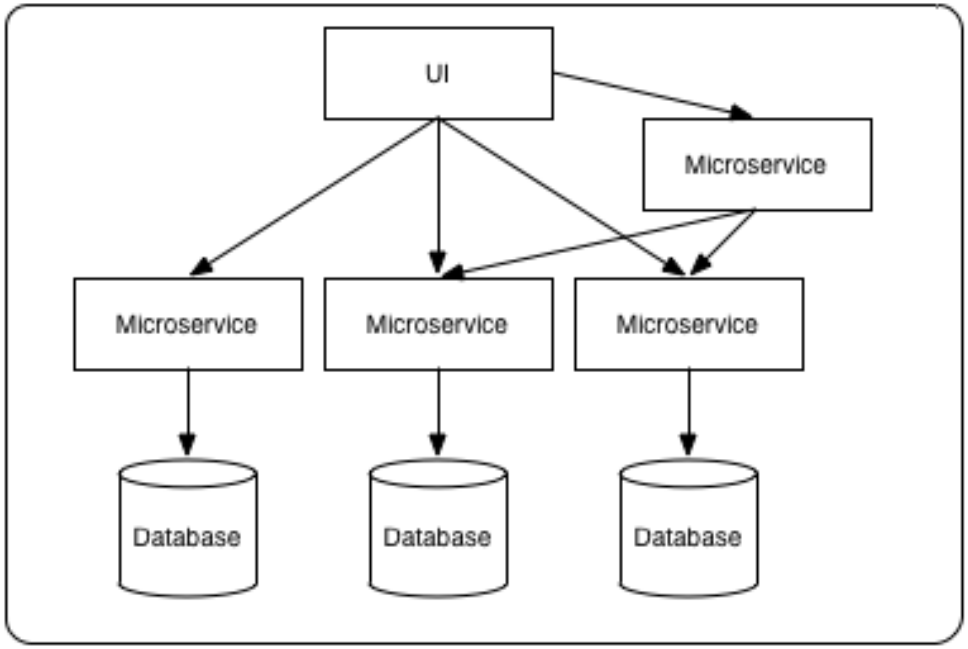


Microservices

Jheronimus Academy of Data Science

Architecture Principles:

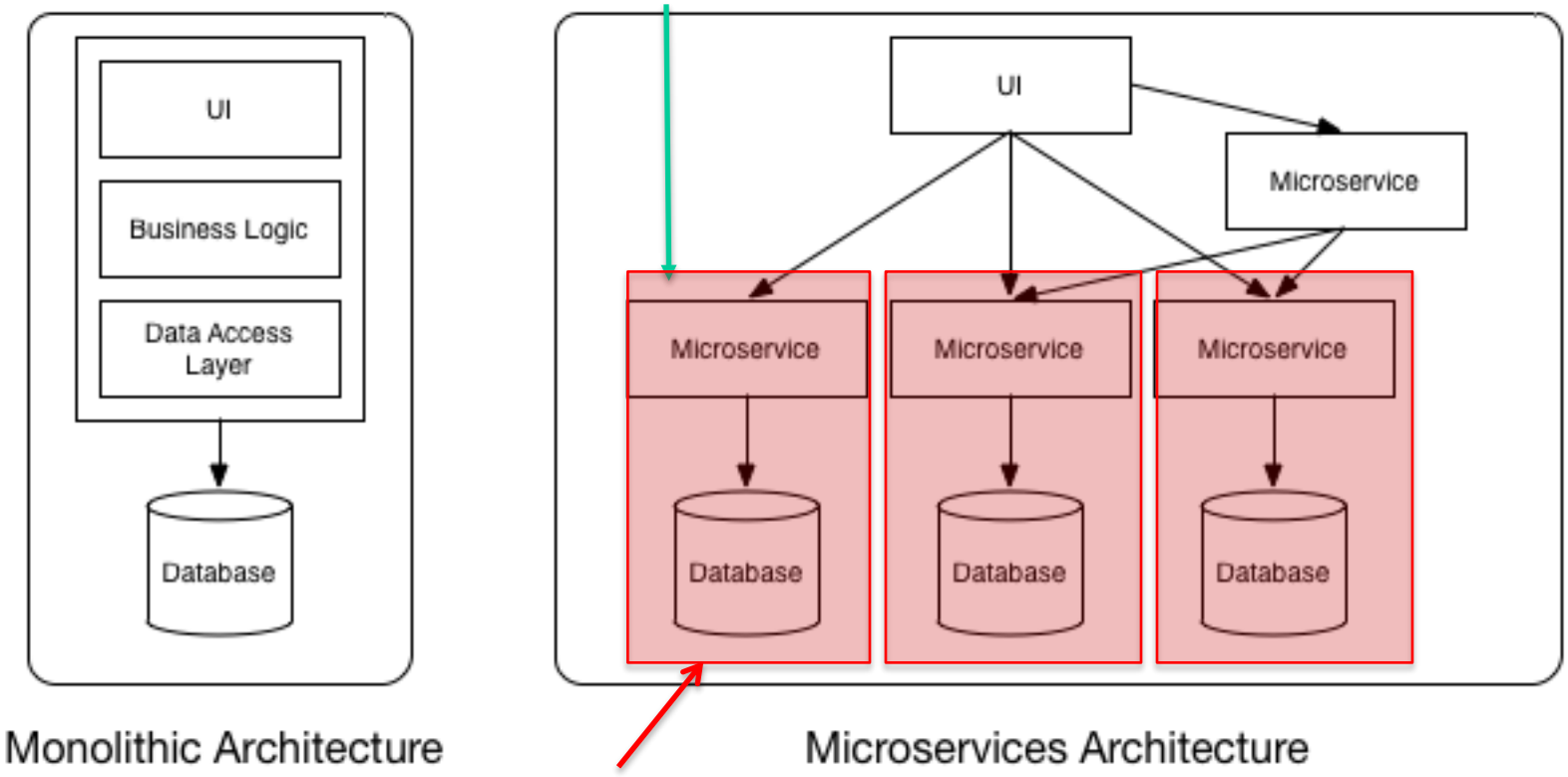
- Statelessness
- Available in multiple copies
- No data sharing among peers
- Each microservice its own DB
- Data-Intensive Message Brokering
  - Message queues



Microservices Architecture

Why Micro-?

Stateless
Available in multiple
No data sharing among microservices
Each microservice uses a specific database



Container!

Maintenance Challenges

- Maintain IaC
- Maintain containerization tech
- Maintain infrastructure back-end (if owned) or go serverless!

Intermezzo: Containerization!

Jheronimus Academy of Data Science

- Terms & Definitions
- Examples & Use-cases

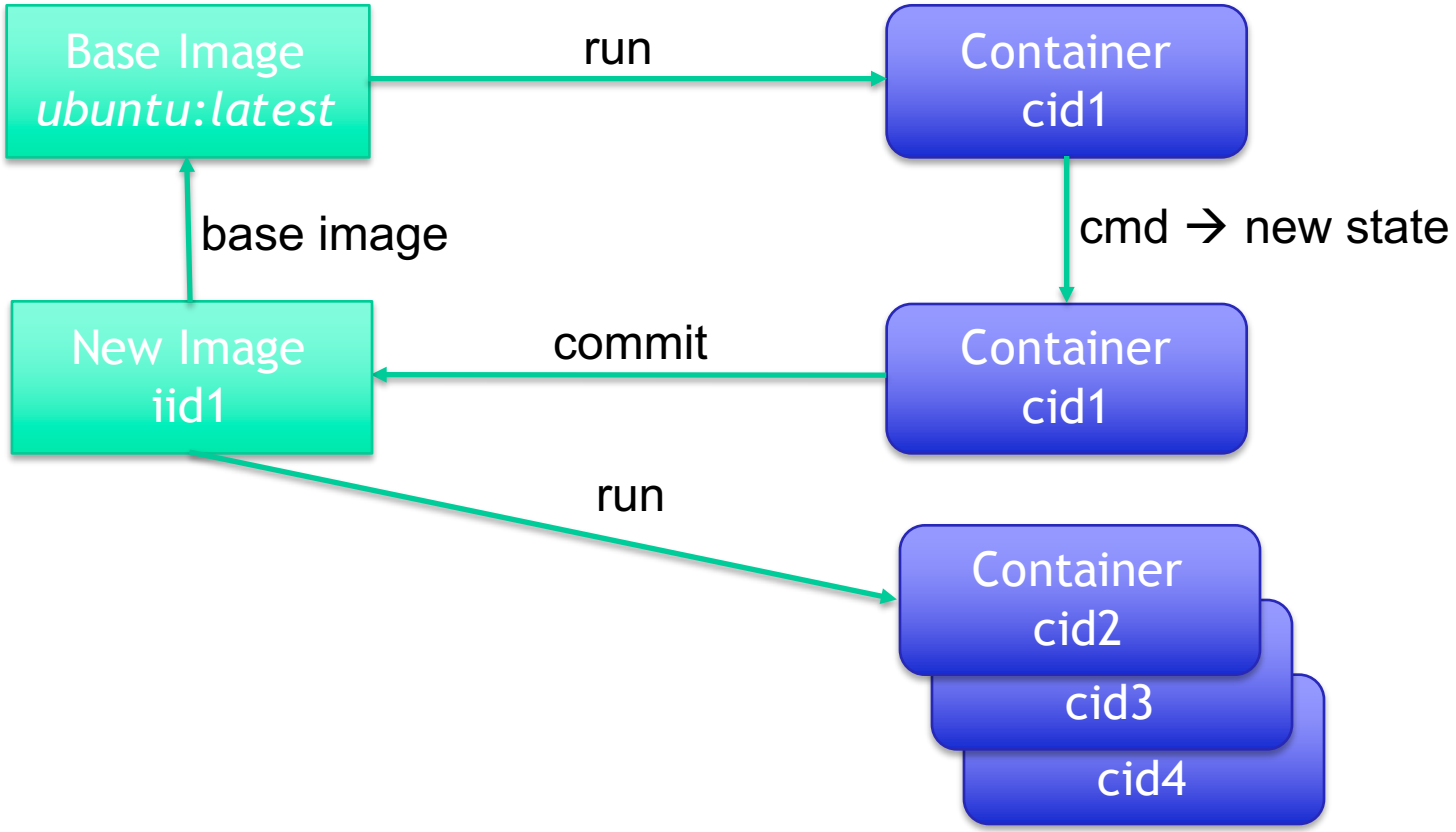
Terminology - Image

- Persisted snapshot that can be run
  - *images:* List all local images
  - *run*: Create a container from an image and execute a command in it
  - *tag*: Tag an image
  - *pull*: Download image from repository
  - *rmi*: Delete a local image
    - This will also remove intermediate images if no longer used

Terminology - Container

- Runnable instance of an image
  - *ps:* List all running containers
  - *ps -a*: List all containers (incl. stopped)
  - *top*: Display processes of a container
  - *start*: Start a stopped container
  - *stop*: Stop a running container
  - *pause*: Pause all processes within a container
  - *rm*: Delete a container
  - *commit*: Create an image from a container

Image vs. Container

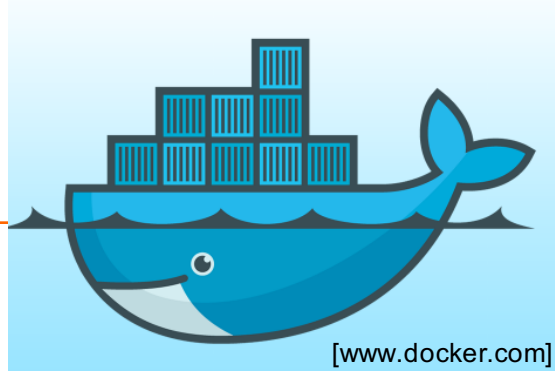


Example: Docker

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system-level virtualization on Linux.

[Source: en.wikipedia.org]

Docker: Name

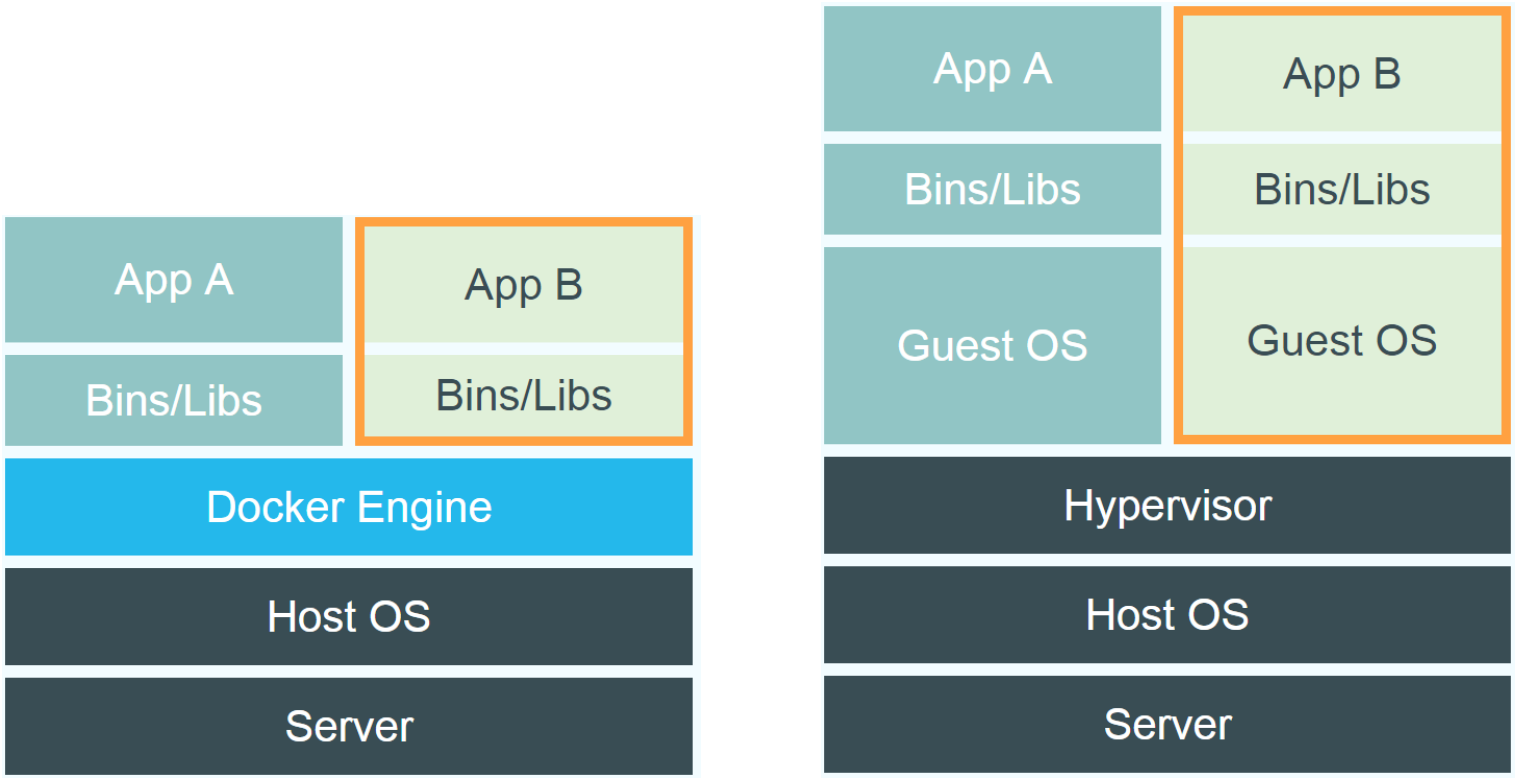


docker [naut.]: der Dockarbeiter, der Hafenarbeiter

Source: leo.org

- Provide a uniformed wrapper around a software package: «*Build, Ship and Run Any App, Anywhere*» [www.docker.com]
  - Similar to shipping containers: The container is always the same, regardless of the contents and thus fits on all trucks, cranes, ships, ...

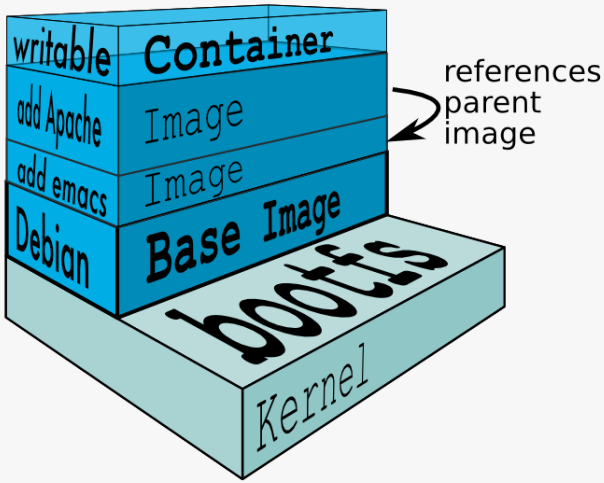
Docker vs. Virtual Machines



Source: https://www.docker.com/whatisdocker/

Docker Technology

- libvirt: Platform Virtualization
- LXC (LinuX Containers): Multiple isolated Linux systems (containers) on a single host
- Layered File System



[Source: https://docs.docker.com/terms/layer/]

Run Platforms

- Various Linux distributions (Ubuntu, Fedora, RHEL, Centos, openSUSE, ...)
- Cloud (Amazon EC2, Google Compute Engine, Rackspace)
- 2014-10: Microsoft announces plans to integrate Docker with next release of Windows Server

Hello World

Jheronimus Academy of Data Science

Simple Command - Ad-Hoc Container

- `docker run ubuntu echo Hello World`
  - `docker images [-a]`
  - `docker ps -a`

Container ➔ Dockerfile

Jheronimus Academy of Data Science

- Create images automatically using a build script: «Dockerfile»
- Can be versioned in a version control system like Git or SVN, along with all dependencies
- Docker Hub can automatically build images based on dockerfiles on Github

Dockerfile Example

Jheronimus Academy of Data Science

- Dockerfile:
  - ```
    FROM ubuntu
    ENV DOCK_MESSAGE Hello My World
    ADD dir /files
    CMD ["bash", "someScript"]
    ```
- `docker build [DockerFileDir]`
- `docker inspect [imageId]`
    ```

Docker Hub

Jheronimus Academy of Data Science

- Public repository of Docker images
  - https://hub.docker.com/
  - docker search [term]
- Automated: Has been automatically built from Dockerfile
  - Source for build is available on GitHub

Docker Use-Cases

Jheronimus Academy of Data Science

- Development Environment
- Environments for Integration Tests
- Quick evaluation of software
- Microservices
- Multi-Tenancy
- Unified execution environment (dev → test → prod (local, VM, cloud, ...)

Docker Use-Cases

Jheronimus Academy of Data Science

- Development Environment
- Environments for Integration Tests
- Quick evaluation of software
- Microservices
- Multi-Tenancy
- Unified execution environment (dev → test → prod (local, VM, cloud, ...)

Hold on… what is *serverless*?

- And why the H*** do I need it?

Basic Idea

- You take a *microservice* and reduce it to a single *function!*

- **Design Principles**
  - **Compute Atomicity**: raw compute-power on demand;
  - **Circa-Zero provisioning:** design for computations, forget about servers;
  - **Trigger-based, event-driven design:** you configure events, (e.g., API requests, uploads) that trigger your serverless function to execute;

Conclusions

- DevOps is a set of tactics that accellerate lead-time;
- We saw two tactics more in depth but there are plenty more and they all cause trouble!
  - Infrastructure code to automate entire delivery pipelines ➔ ***maintenance nightmare!***
  - Microservices & containers + serverless granularize the architecture ➔ ***maintenance nightmare!***
- Research and Practice around DevOps are converging;

Example Senlin "scaling_out_policy_ceilometer.yaml"

using the Kubernetes "redis" example

Jheronimus Academy of Data Science

Target is a Kubernetes Pod of the tosca.groups.placement type

Symbolic name for the trigger *(could be used to reference an externalized version; however, this would violate a Policy's integrity as a "Security document"*

TOSCA Policy

```
my_scaling_policy:
  type: tosca.policies.scaling
  properties:  # normative TOSCA properties for scaling
    min_instances: 1
    max_instances: 10
    default_instances: 3
    increment: 1
  # target the policy at the "Pod"
  targets: [redis-master-pod]
  triggers:
    resize_compute:  # symbolic name
      event: tosca.events.resource.utilization
      target_filter:
        node: master-container
        requirement: host
        capability: Container
      condition: utilization greater_than 80%
      action:
        # map to SENLIN::ACTION::RESIZE
        scaleup: # logical operation name
          inputs:  # optional inputs parameters
            number: 1
            strategy: BEST_EFFORT
          Implementation: <script> | <service_name>
```

TOSCA normative event type (name) that would map to domain-specific names (e.g., OpenStack Ceilometer)

Find the attribute via the topology:

a) Navigate to node (directly or via the requirement name) and optionally the Capability name

b) The condition to map & register with the target monitoring service (e.g., Ceilometer)

Describe NODE to attach an alarm | alert | event to

i.e., Using the "node", "req", "cap" and "condition" keys would expressed as a descriptive "filter"

TODO:
Need a % data type for TOSCA

List optional input parms. here

Note: we combined the Senlin "Action" of ***SENLIN:ACTION:RESIZE*** *with the strategy:* ***BEST_EFFORT*** *to have one name*

more info online

That’s all folks!

Jheronimus Academy of Data Science

Any Questions?

References

[1] Erder, Murat and Pureur, Pierre. Continuous Architecture: Sustainable Architecture in an Agile and Cloud-Centric World. Amsterdam: Morgan Kaufmann, 2016.

[2] Continuous Testing Paperback – January 2, 2014 by W. Ariola, C. Dunlop

[3] Part of the Pipeline: Why Continuous Testing Is Essential, by Adam Auerbach, TechWell Insights August 2015

[4] M. Fowler Continuous Integration, https://www.thoughtworks.com/continuous-integration

[5] *Chen, Lianping (2015) "Continuous Delivery: Huge Benefits, but Challenges Too" IEEE Software. **32** (2): 50.*

[6] http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd03/TOSCA-Simple-Profile-YAML-v1.0-csd03.html

[7] P. Lipton, D. Palma, M. Rutkowski, and D. A. Tamburri, "Tosca solves big problems in the cloud and beyond!" *IEEE Cloud*, vol. 21, no. 11, pp. 31–39, 2016.

[8] Bengtsson, PerOlof, Lassing, Nico, Bosch, Jan and van Vliet, Hans. "Architecture-level modifiability analysis (ALMA)." *Journal of Systems and Software* 69 , no. 1--2 (2004): 129--147.

[9] Tamburri, D. A.; Lago, P. & van Vliet, H. (2013), 'Uncovering Latent Social Communities in Software Development.', *IEEE Software* **30** (1) , 29-36 .

[10] M. Di Penta, D. A. Tamburri, Combining Quantitative and Qualitative Methods in Empirical Software Engineering Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM Sigsoft Symposium of the Foundations of Software

Other Biblio

[11] Bass, L. J.; Weber, I. M. & Zhu, L. (2015), *DevOps - A Software Architect's Perspective*. , Addison-Wesley .

[12] Tamburri, D. A. & Nitto, E. D. (2015), When Software Architecture Leads to Social Debt., *in* Len Bass; Patricia Lago & Philippe Kruchten, ed., 'WICSA' , IEEE Computer Society, pp. 61-64 .

[13] Tamburri, D. A.; Kruchten, P.; Lago, P. & van Vliet, H. (2015), 'Social debt in software engineering: insights from industry.', *J. Internet Services and Applications* **6** (1) , 10:1-10:17 .

[14] Tamburri, D. A.; Lago, P. & van Vliet, H. (2013), 'Organizational social structures for software engineering.', *ACM Comput. Surv.* **46** (1) , 3 .