RADON: A Story of One Serverless Journey

Slides courtesy: RADON Consortium

Vladimir Yussupov, *University of Stuttgart*

Agenda

1 RADON – Motivation & Overview

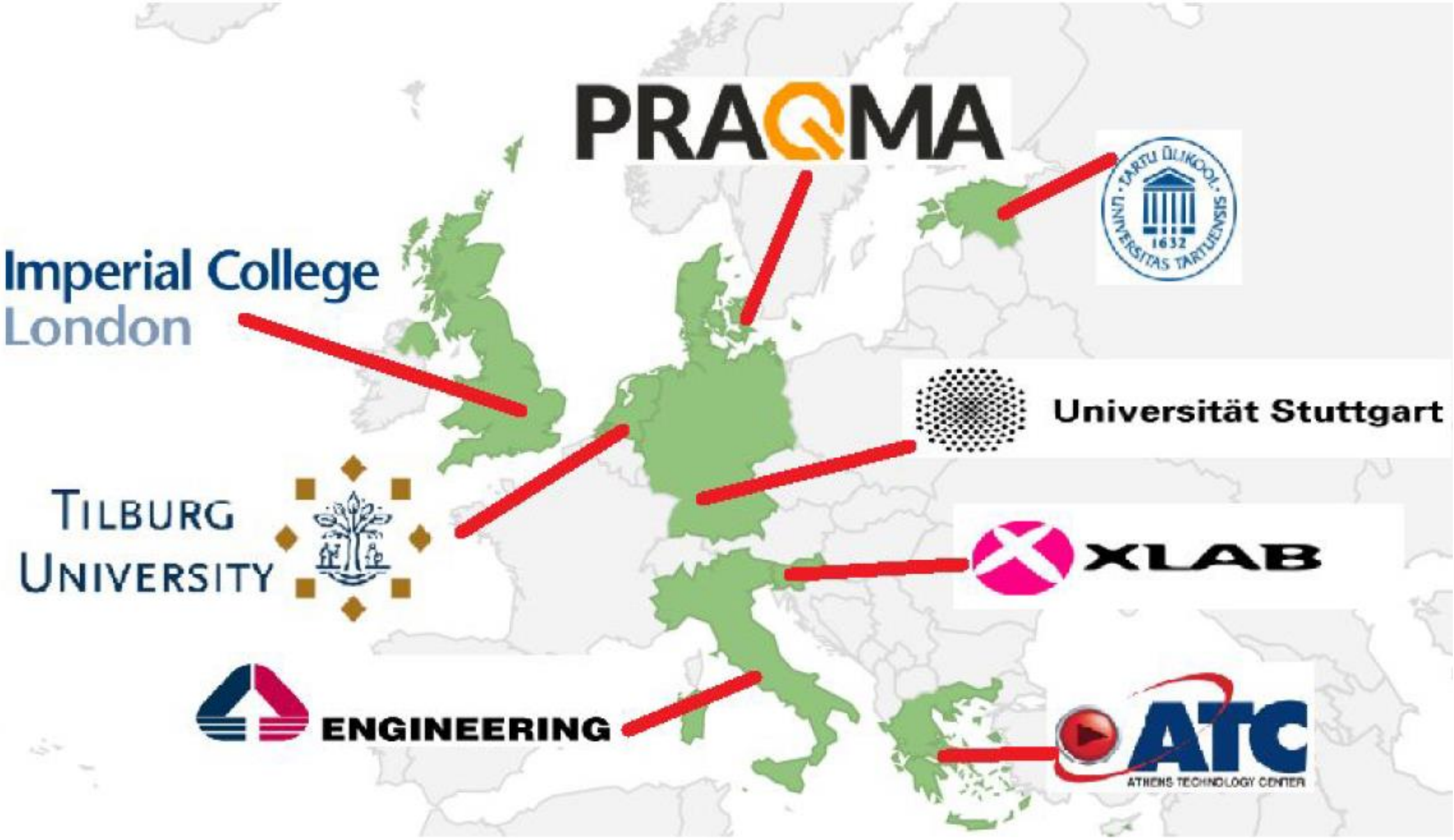
2 TOSCA & (Graphical) Modeling

3 RADON Quality Assurance Tools

4 RADON Runtime Environment & IDE

Rational Decomposition and Orchestration for Serverless Computing

- 30 months EU H2020 project, 8 organizations (completed in June 2021)
- **Value Proposition**: a **DevOps framework** to help the EU software industry adopting **serverless FaaS** without vendor lock-in



Serverless Computing & Function-as-a-Service

Often associated with the term “serverless”

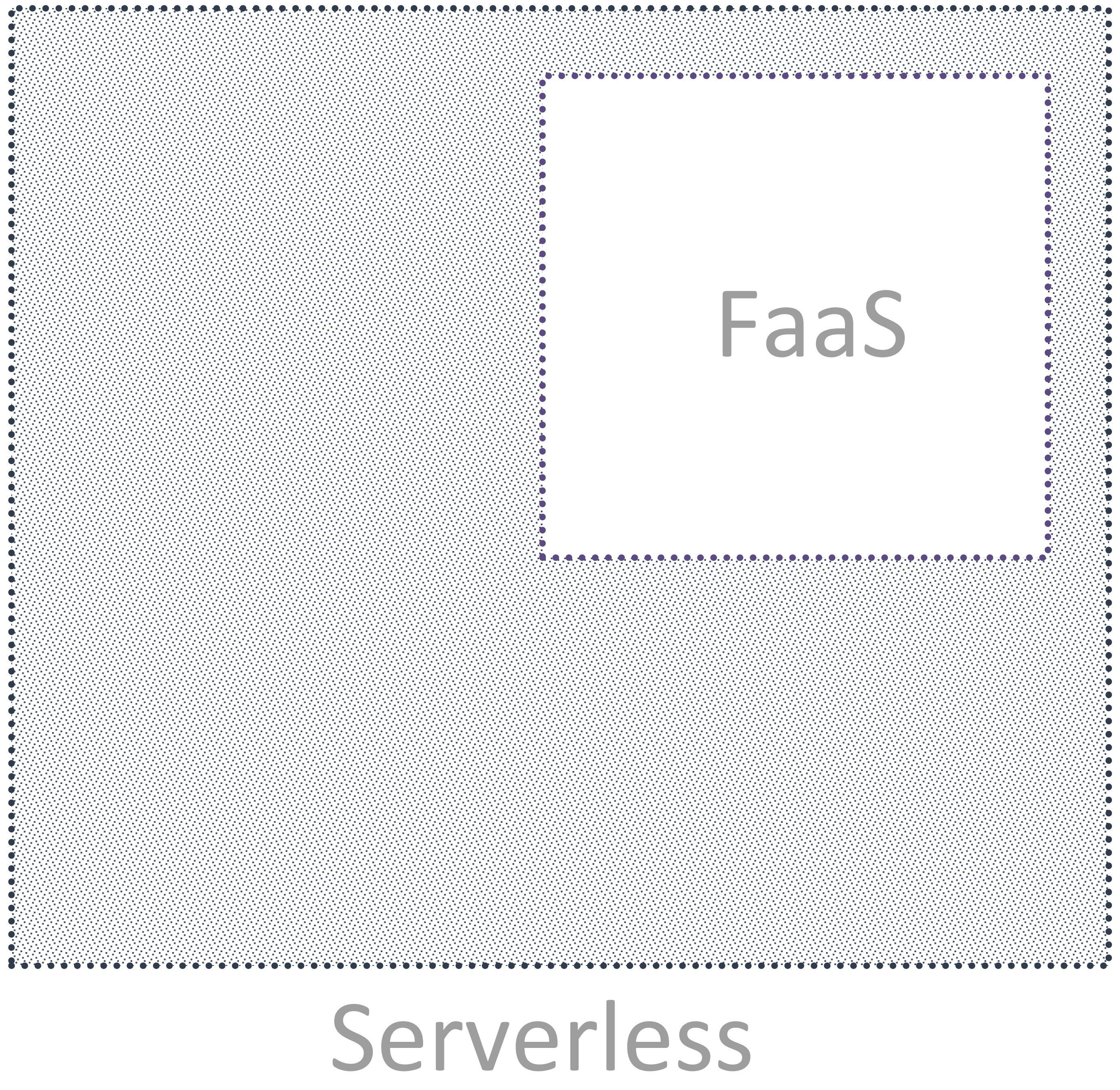
- Function-as-a-Service (FaaS) offerings
  - AWS Lambda, Azure Functions, OpenFaaS, …
- FaaS execution model

What serverless also encompasses

- FaaS / DBaaS / Message Queues as a Service / SaaS / …
- focus: shift management efforts on providers

Core differences from serverful computing*

- Compute / storage are provisioned & priced separately
  - separate services, computation is stateless, …
- Automated, provider-managed resources allocation
- Billing associated with execution



* Jonas et al. “Cloud Programming Simplified: A Berkeley View on Serverless Computing”

Function-as-a-Service (FaaS) Processing Model

- Fine-grained functions hosted in the cloud and fully managed by the provider
- Cost-savings in event-driven workloads (e.g., IoT)
- Strong synergy with microservices
- Resource decoupling
  - stateless functions
  - state persisted via storage
  - state change can produce events

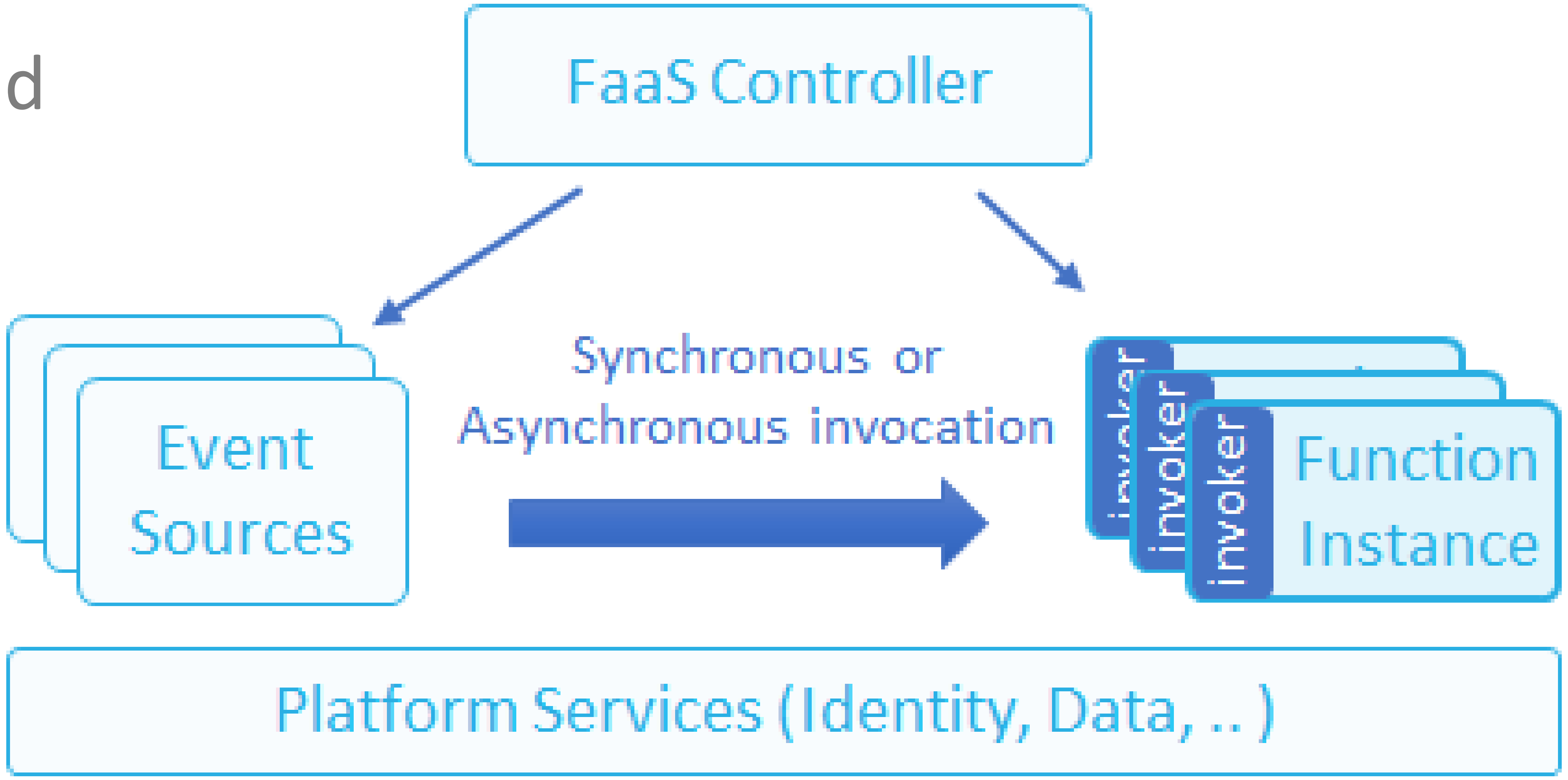
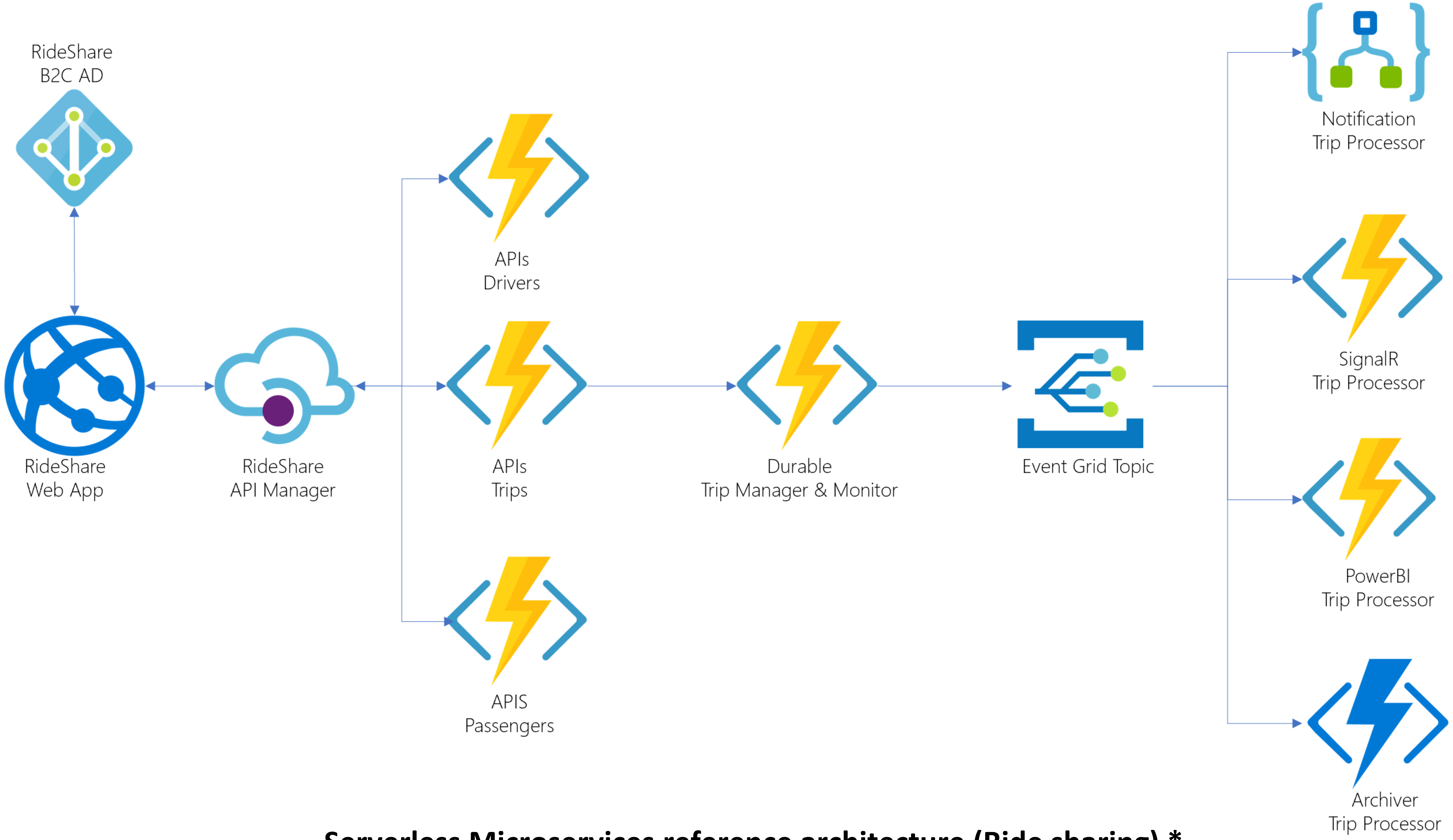

FaaS Controller
Event Sources
Synchronous or Asynchronous invocation
invoker
invoker
invoker
Function Instance
Platform Services (Identity, Data, ..)

Illustration from the CNCF Serverless White Paper

Serverless Architectures & Microservices



Serverless Microservices reference architecture (Ride sharing) *

* From https://docs.microsoft.com/en-us/samples/azure-samples/serverless-microservices-reference-architecture/serverless-microservices-reference-architecture/

Why serverless/FaaS is a key software technology?

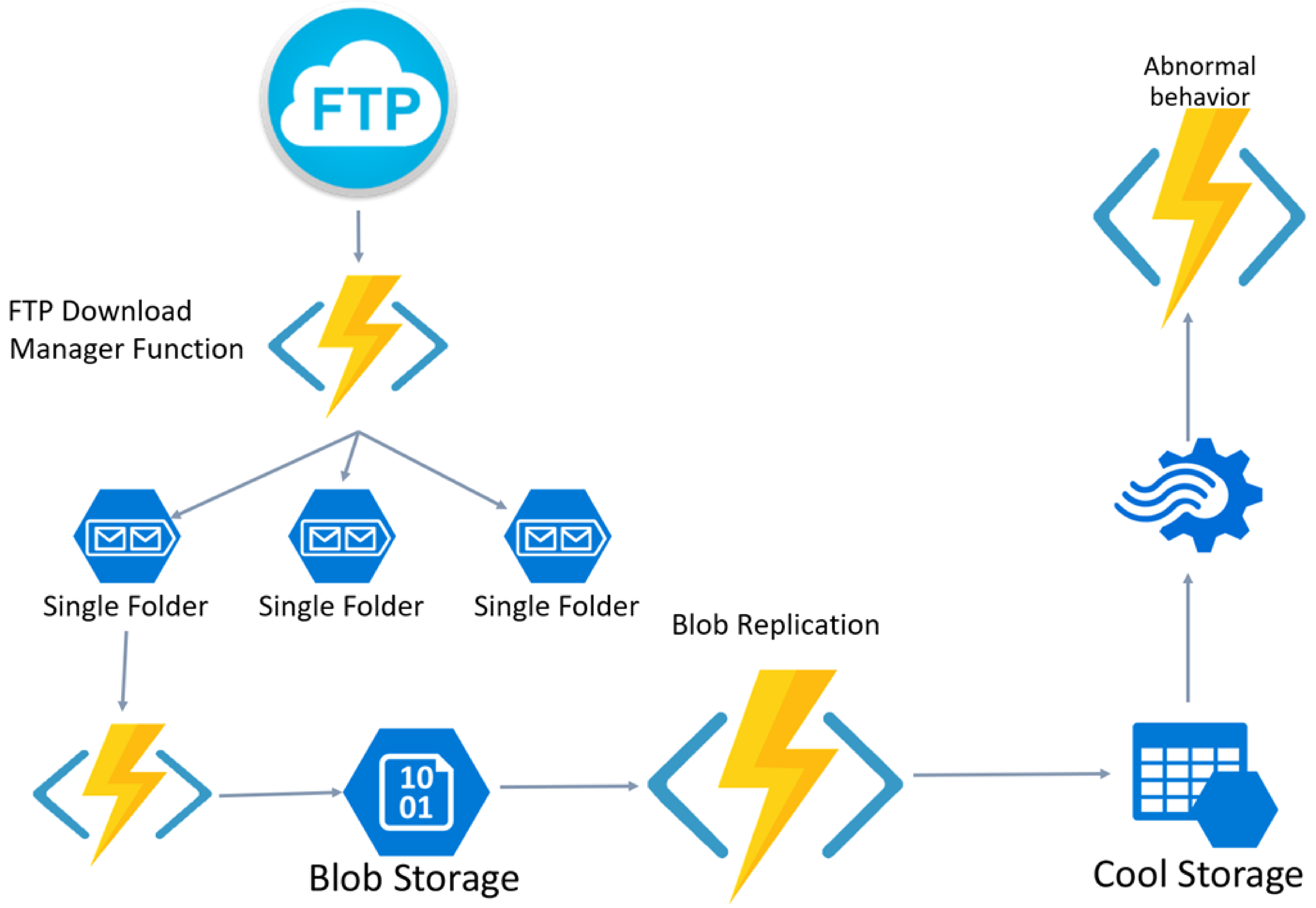
- A way to quickly prototype cloud-based applications
  - Customer demonstration without infrastructure management issues
  - In some cases, demos can be built in hours
- A way to avoid unwarranted costs
  - Functions are deallocated automatically
  - Risk reduction for SME/startups
- Natural to combine with microservices-based architectures
  - Fine-grained software architecture
  - Automated autoscaling
  - Flexibility and responsiveness
  - High-degree of reuse of platform services

Various application domains

- Real-time data analytics & file processing
- Serverless APIs
- Periodic function invocations
- Batch processing, Map-Reduce
- IoT, e.g. using FaaS to connect devices with end-users through cloud
- Financial data analytics as data processing of transactions for insider trading
- Managing accounts and trading actions
- Serving static content/websites
- Extract, transform, load data

…

Serverless Architecture Examples



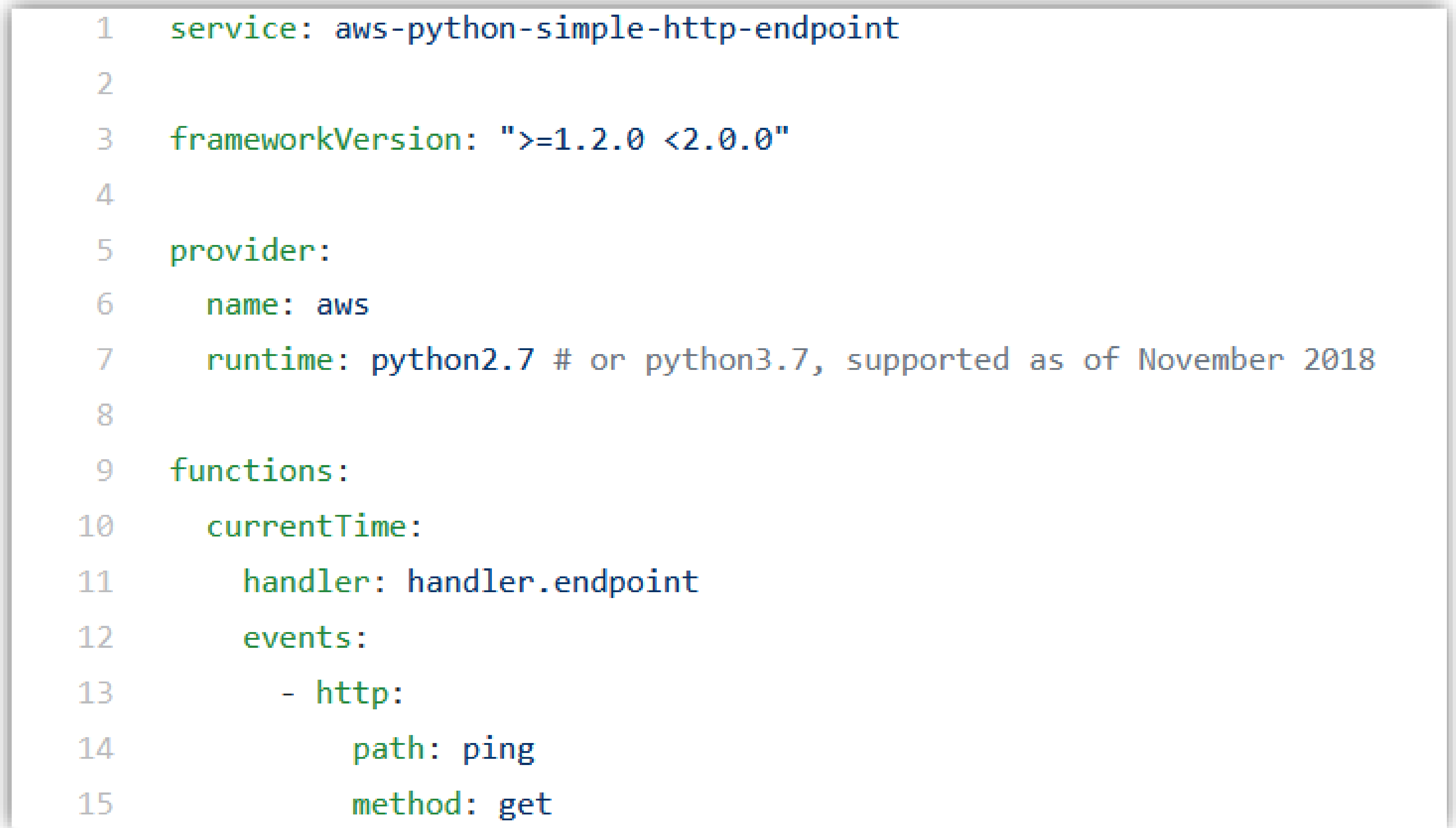
Serverless ETL Pipeline*

*From https://docs.microsoft.com/en-us/dotnet/architecture/serverless/serverless-design-examples

Problem: (Models) Heterogeneity

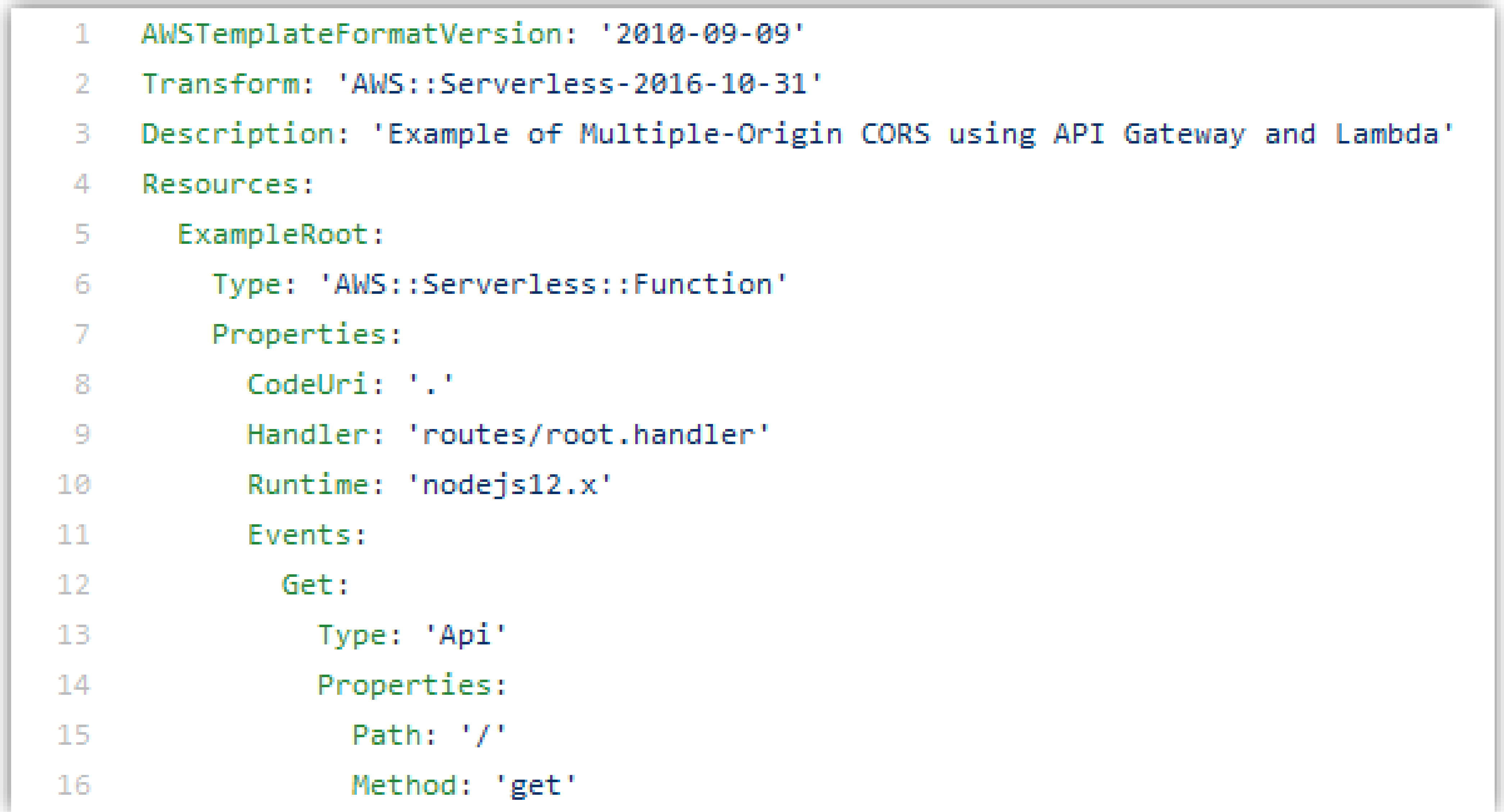
puppet
labs
δ·CLOUD
serverless
CHALICE
Bash Shell
juju
openstack™
libcloud
Node Lambda
CFEngine
SmartFrog
kubernetes
CloudFormation
heat
ANSIBLE
CLOUD
FOUNDRY™
VAGRANT
Cloudify
docker

Deployment Models Example



```yaml
service: aws-python-simple-http-endpoint

frameworkVersion: ">=1.2.0 <2.0.0"

provider:
  name: aws
  runtime: python2.7 # or python3.7, supported as of November 2018

functions:
  currentTime:
    handler: handler.endpoint
    events:
      - http:
          path: ping
          method: get
```

Deploy HTTP Endpoint to AWS
using Serverless Framework



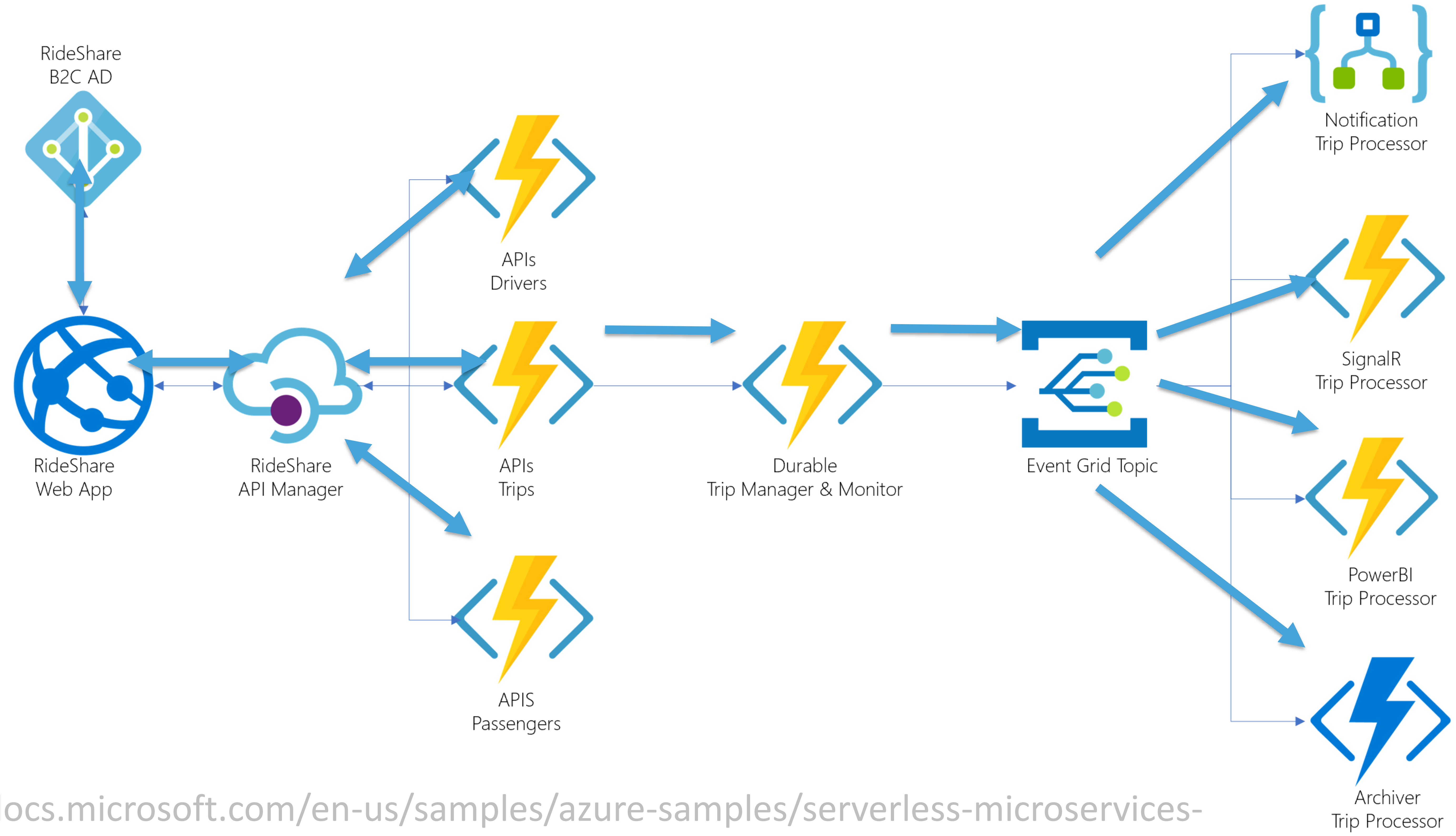
```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: 'Example of Multiple-Origin CORS using API Gateway and Lambda'
Resources:
  ExampleRoot:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: '.'
      Handler: 'routes/root.handler'
      Runtime: 'nodejs12.x'
      Events:
        Get:
          Type: 'Api'
          Properties:
            Path: '/'
            Method: 'get'
```

Deploy HTTP Endpoint to AWS
using AWS SAM

Modeling Not Only for Deployment Automation

An executable deployment model contains valuable details about:

- relations among components (connectivity)
- their types (function, storage, message queue, …)

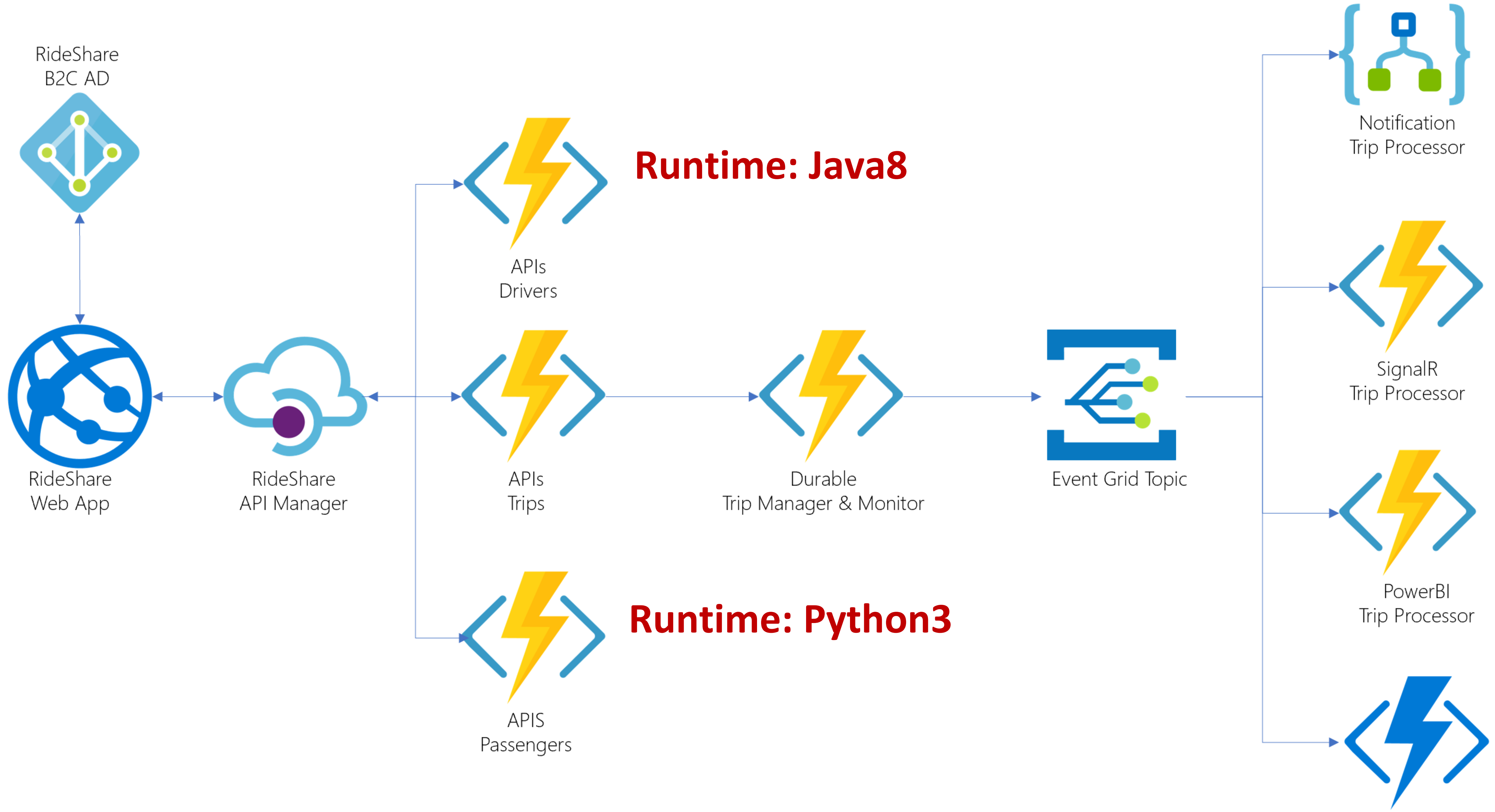


* From https://docs.microsoft.com/en-us/samples/azure-samples/serverless-microservices-reference-architecture/serverless-microservices-reference-architecture/

Modeling Not Only for Deployment Automation

An executable deployment model contains valuable details about:

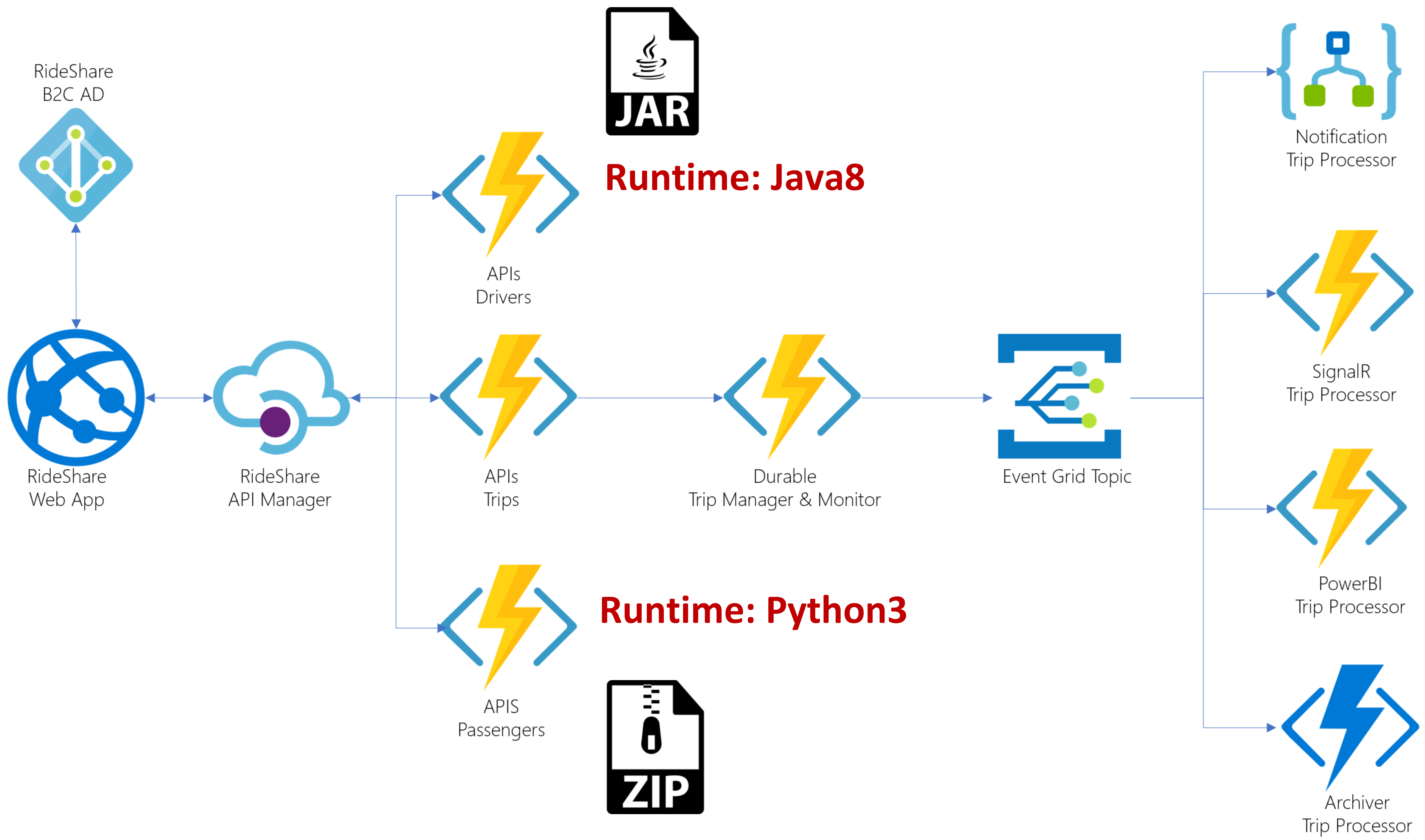
- Component properties and attributes



Modeling Not Only for Deployment Automation

An executable deployment model contains valuable details about:

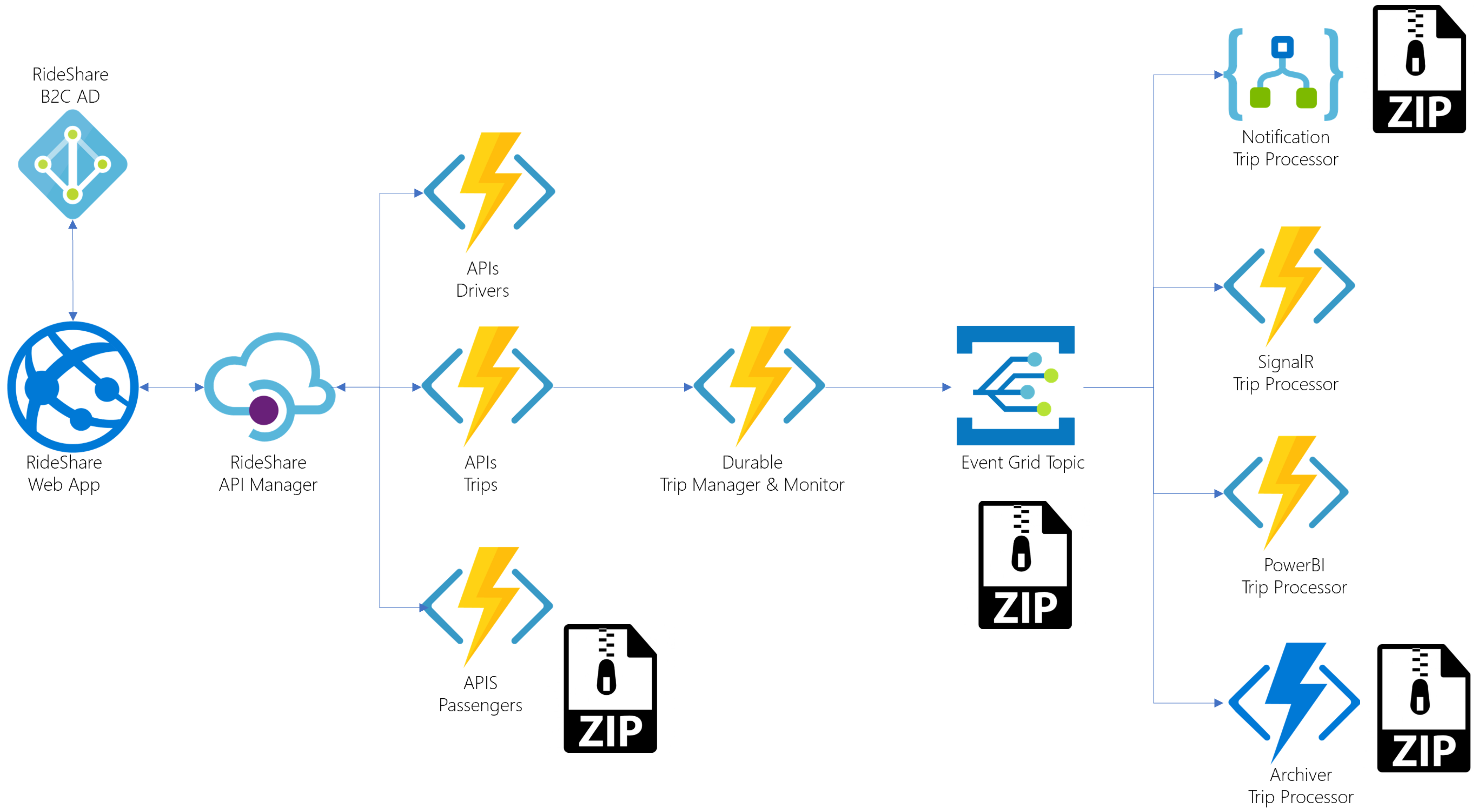
- Components business logic



Modeling Not Only for Deployment Automation

An executable deployment model contains valuable details about:

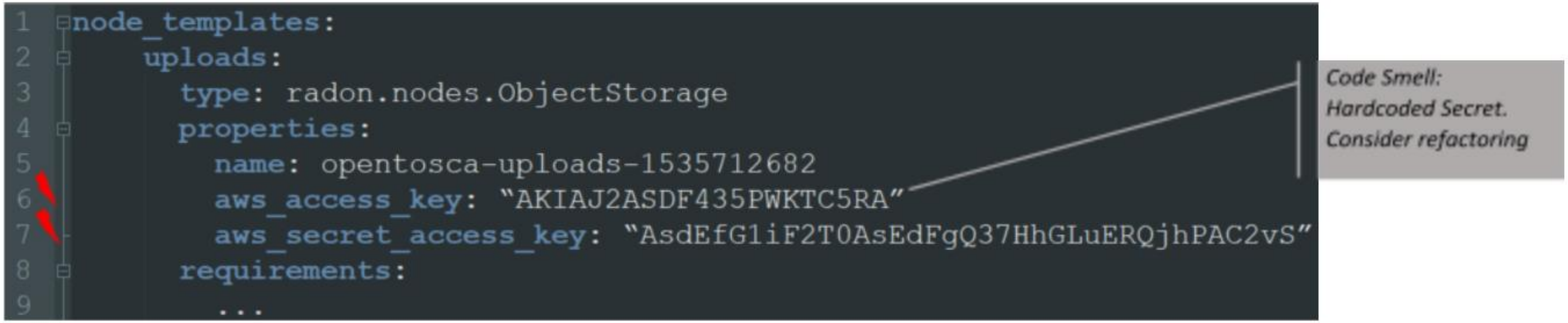
- Component deployment / configuration logic
  - scripts to run before / after deployment, etc.



Modeling Not Only for Deployment Automation

This information can be used:

- Find defects in code, e.g., Anti-patterns, Code smells



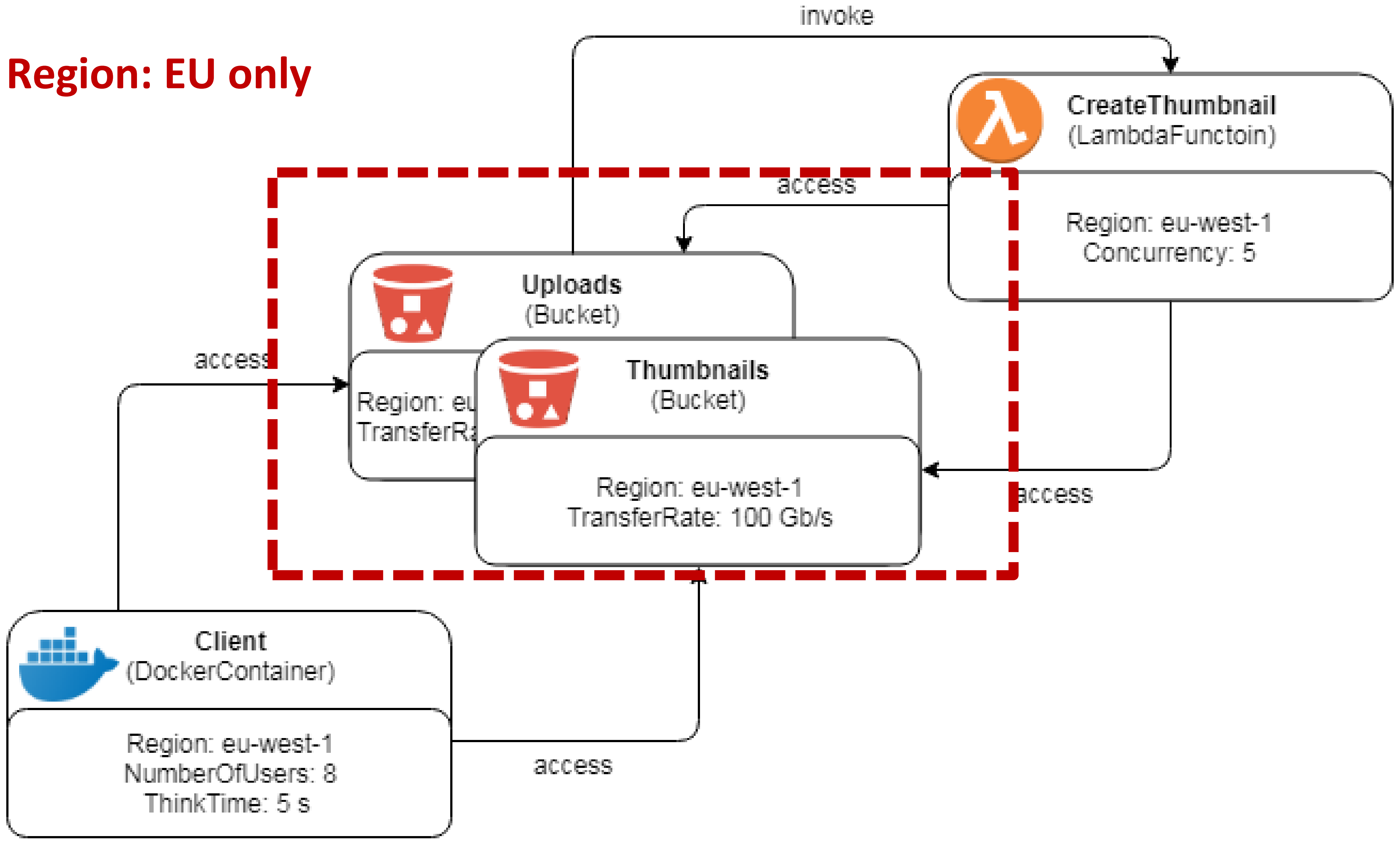
```
node_templates:
    uploads:
      type: radon.nodes.ObjectStorage
      properties:
        name: opentosca-uploads-1535712682
        aws_access_key: "AKIAJ2ASDF435PWKTC5RA"
        aws_secret_access_key: "AsdEfG1iF2T0AsEdFgQ37HhGLuERQjhPAC2vS"
      requirements:
        ...
```

Code Smell:
Hardcoded Secret.
Consider refactoring

Modeling Not Only for Deployment Automation

This information can be used:

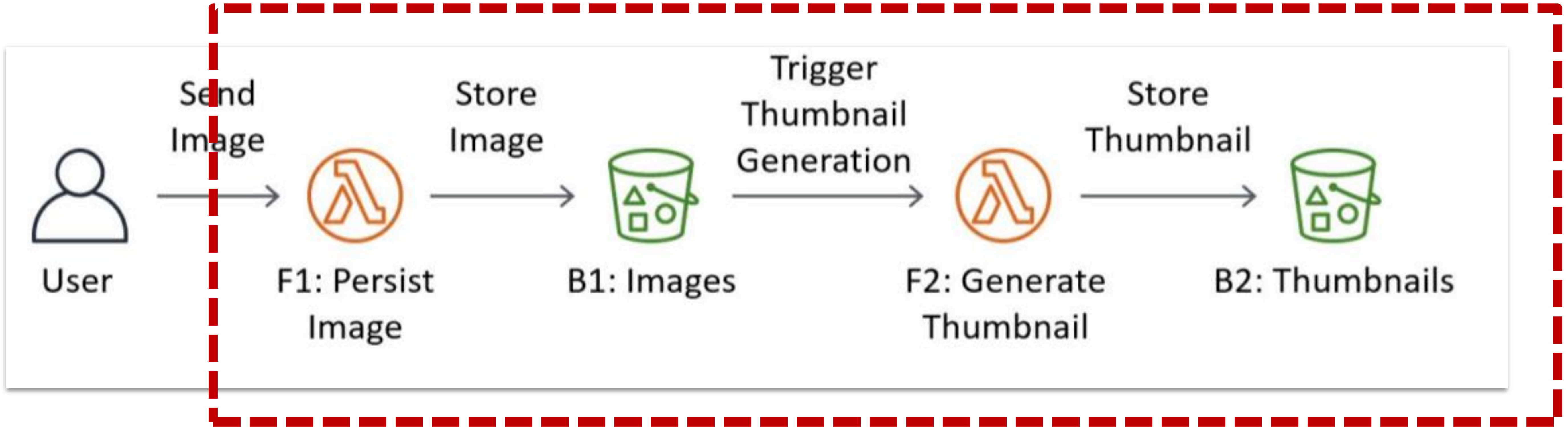
- Verify if some constraints are satisfied
  - All data must reside on EU territory
  - Functions must only interact using API Gateway



Modeling Not Only for Deployment Automation

This information can be used:

- To enable continuous testing
  - Functional (deployment)
  - Non-functional
    - Baseline performance/costs of functions (and their configurations)
    - End-to-end times of function/bucket interactions
  - Levels: unit (function), integration/system (including triggering) || regression

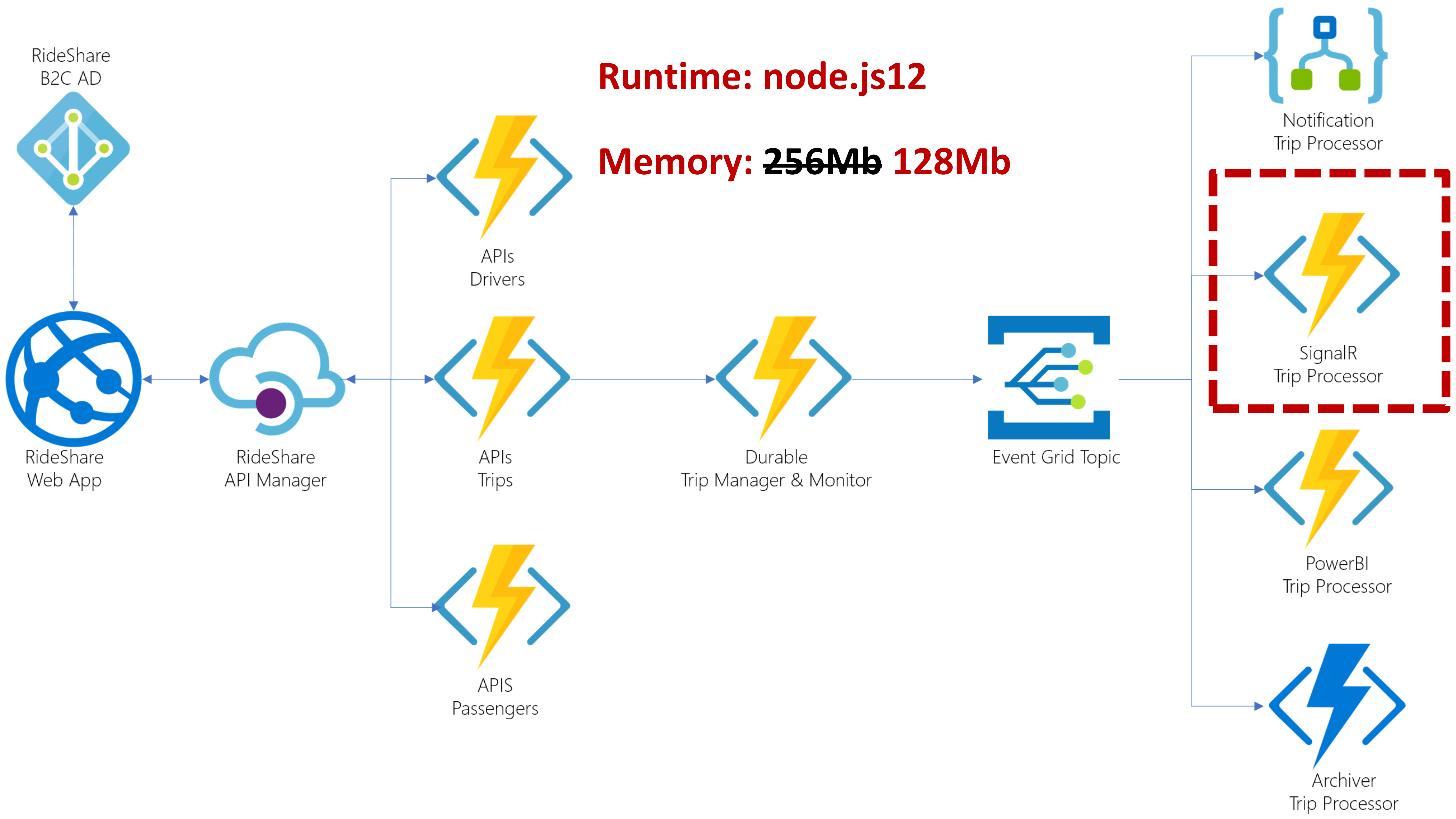


Test:
A thumbnail is generated successfully after deployment

Modeling Not Only for Deployment Automation

This information can be used:

- Optimize deployment, e.g., memory settings based on runtime usage data
- Decompose functionalities into smaller units



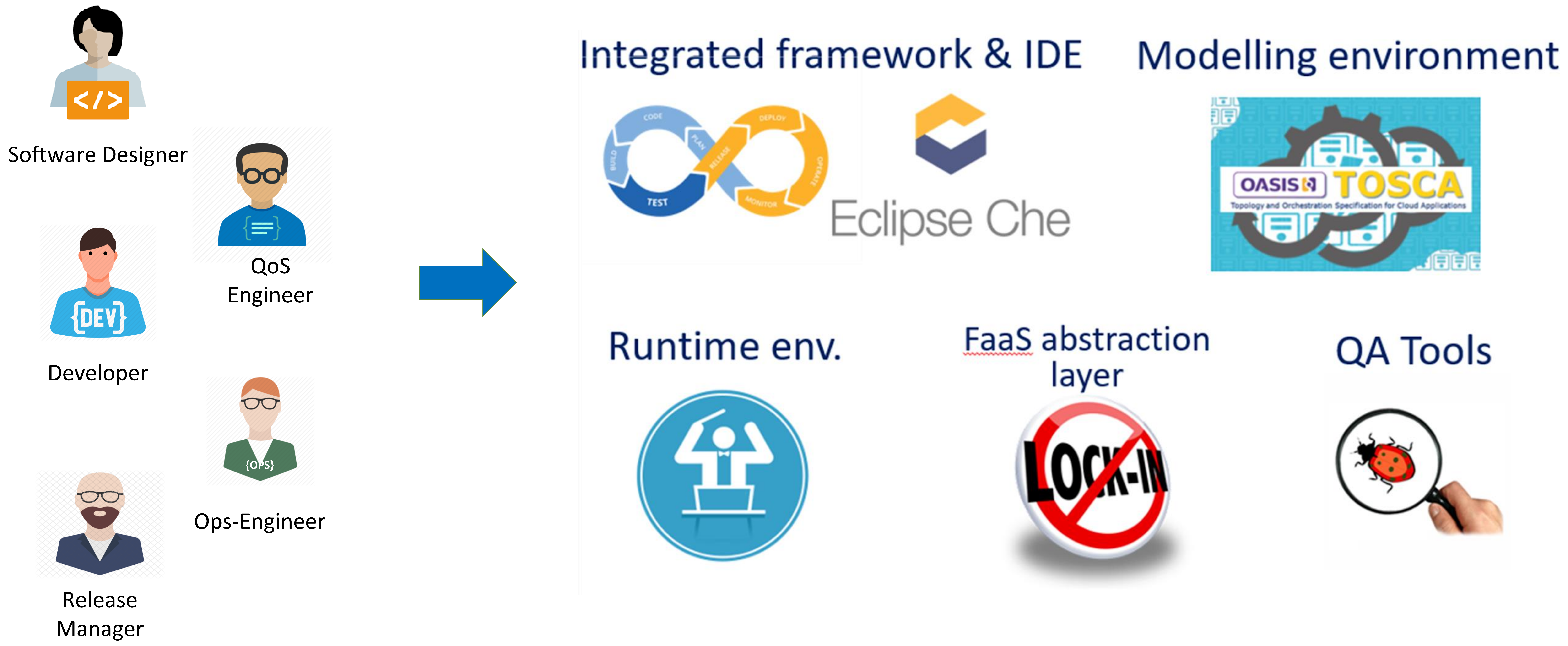
Runtime: node.js12

Memory: ~~256Mb~~ 128Mb

Can be split into 2 microservices!

Now Imagine Supporting These Use Cases for ALL Deployment Technologies!

RADON: Model-driven DevOps Framework



Software Designer

QoS Engineer

Developer

Ops-Engineer

Release Manager

Integrated framework & IDE

Modelling environment

Runtime env.

FaaS abstraction layer

QA Tools

RADON

Brief Introduction to OASIS TOSCA

Standards-based Deployment Modeling: TOSCA 101

- The ***Topology and Orchestration Specification for Cloud Applications (TOSCA):*** an OASIS standard for automating the deployment and management of cloud applications in a portable manner
- The major goals of TOSCA are:
  - Automation of Deployment and Management
  - Portability
  - Interoperability
  - Vendor-neutral ecosystem

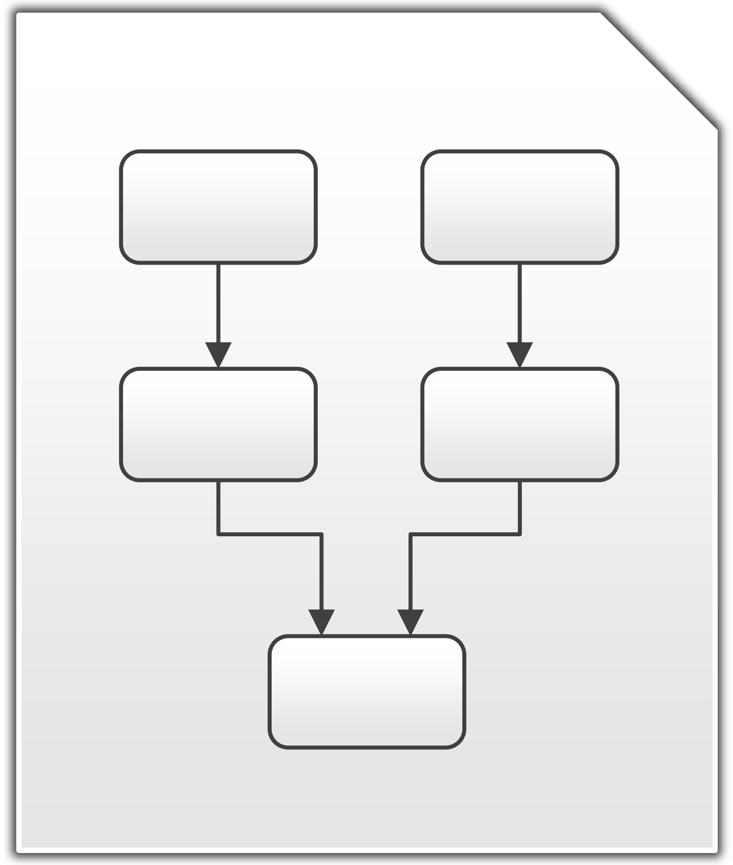
→ **OASIS Awards 2017 Open Standards Cup to TOSCA for Cloud Portability**

Standards-based Deployment Modeling: TOSCA 101

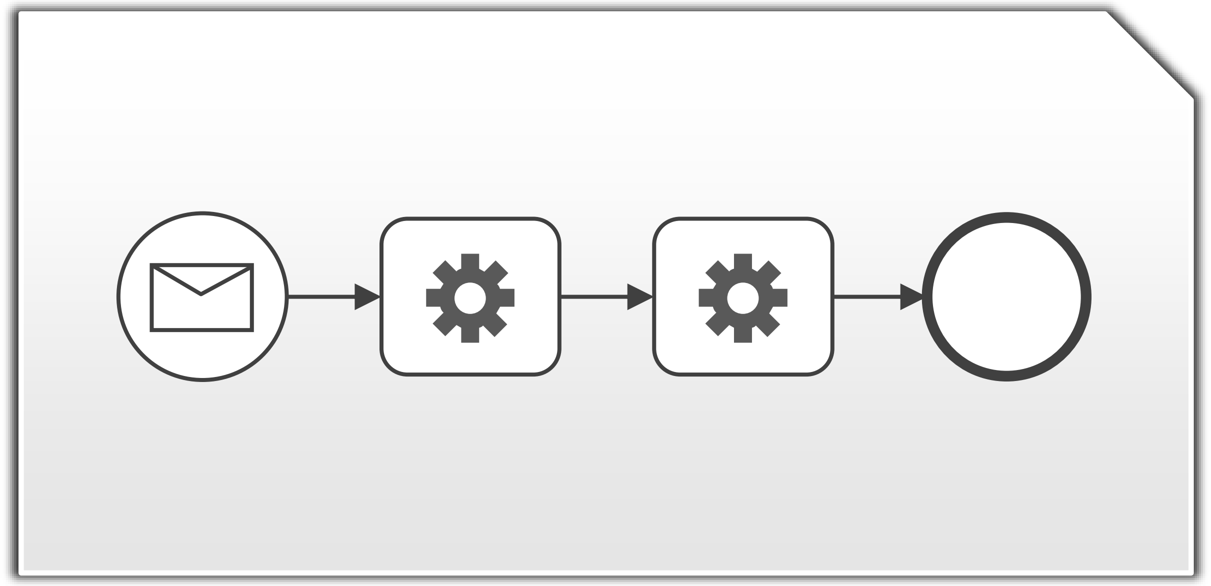
TOSCA:

Topology and **Orchestration**
Specification for Cloud Applications

Application Structure



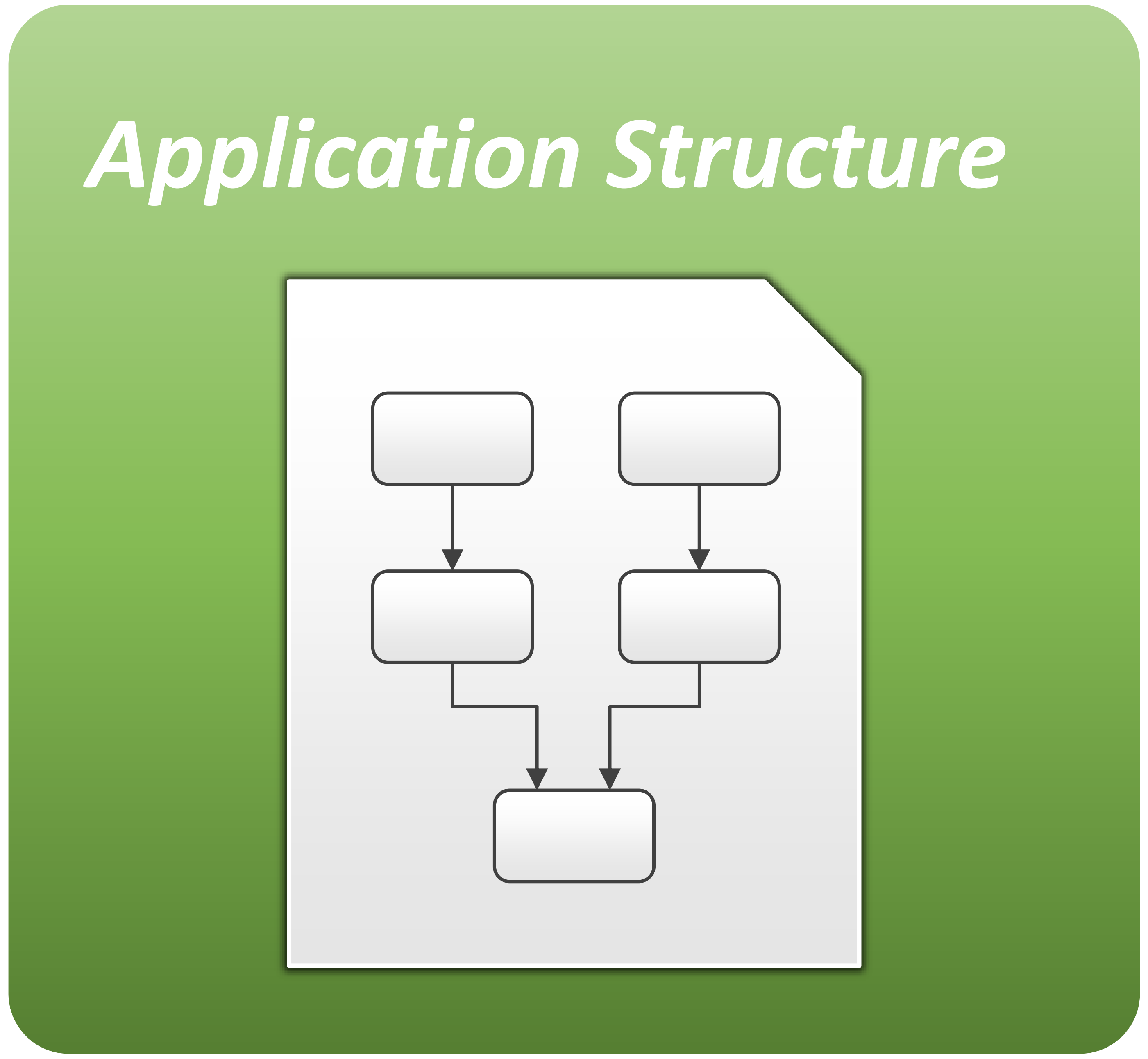
Deployment & Management



Standards-based Deployment Modeling: TOSCA 101

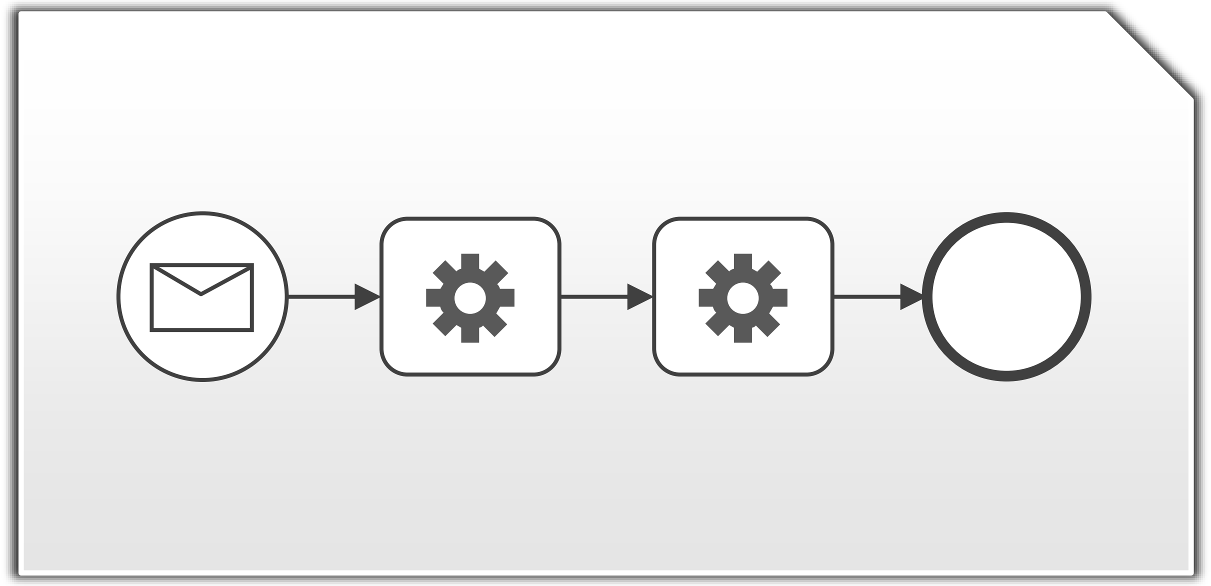
TOSCA:

Topology and **Orchestration**
Specification for Cloud Applications

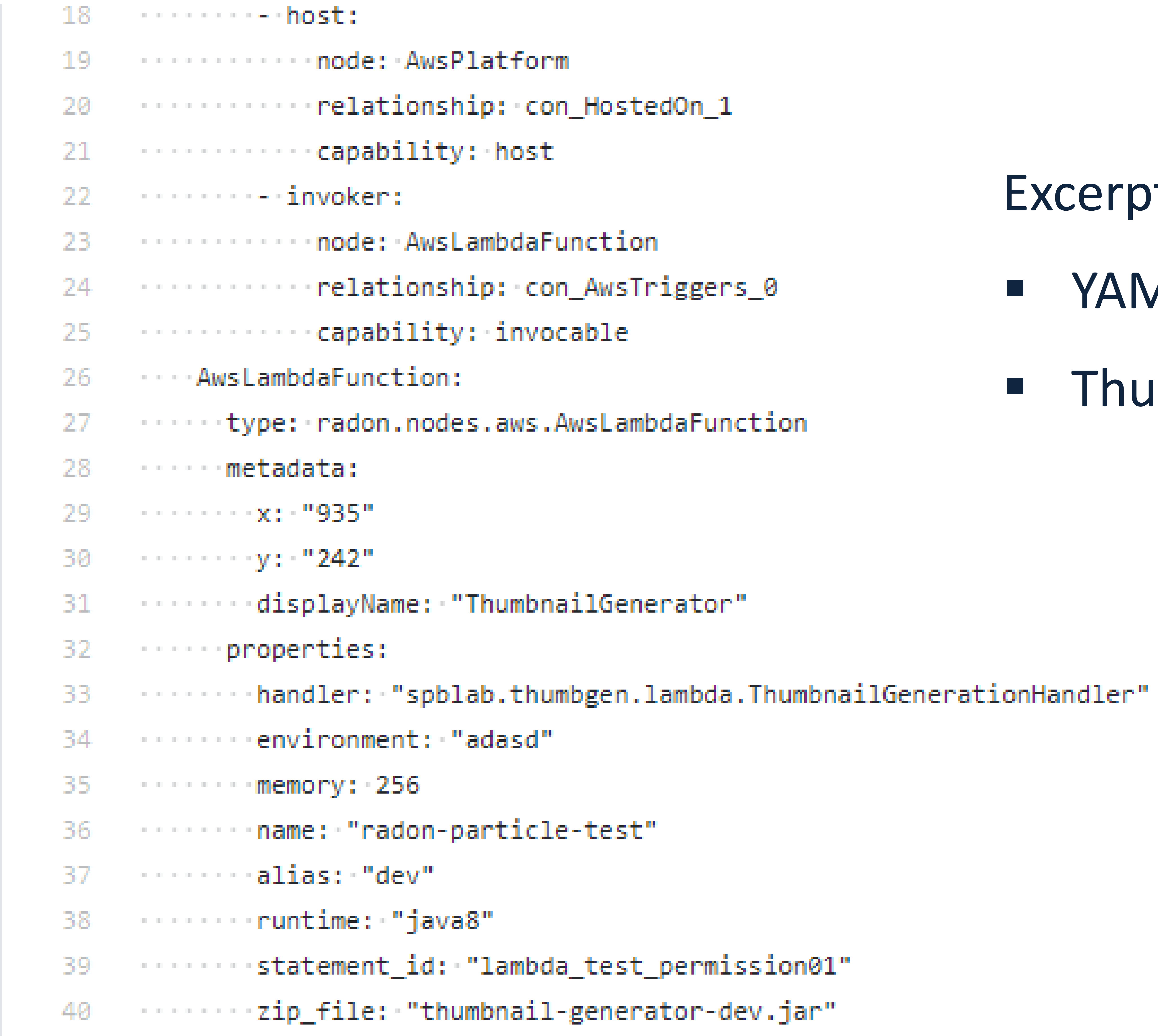


Application Structure

Deployment & Management



Standards-based Deployment Modeling: TOSCA 101

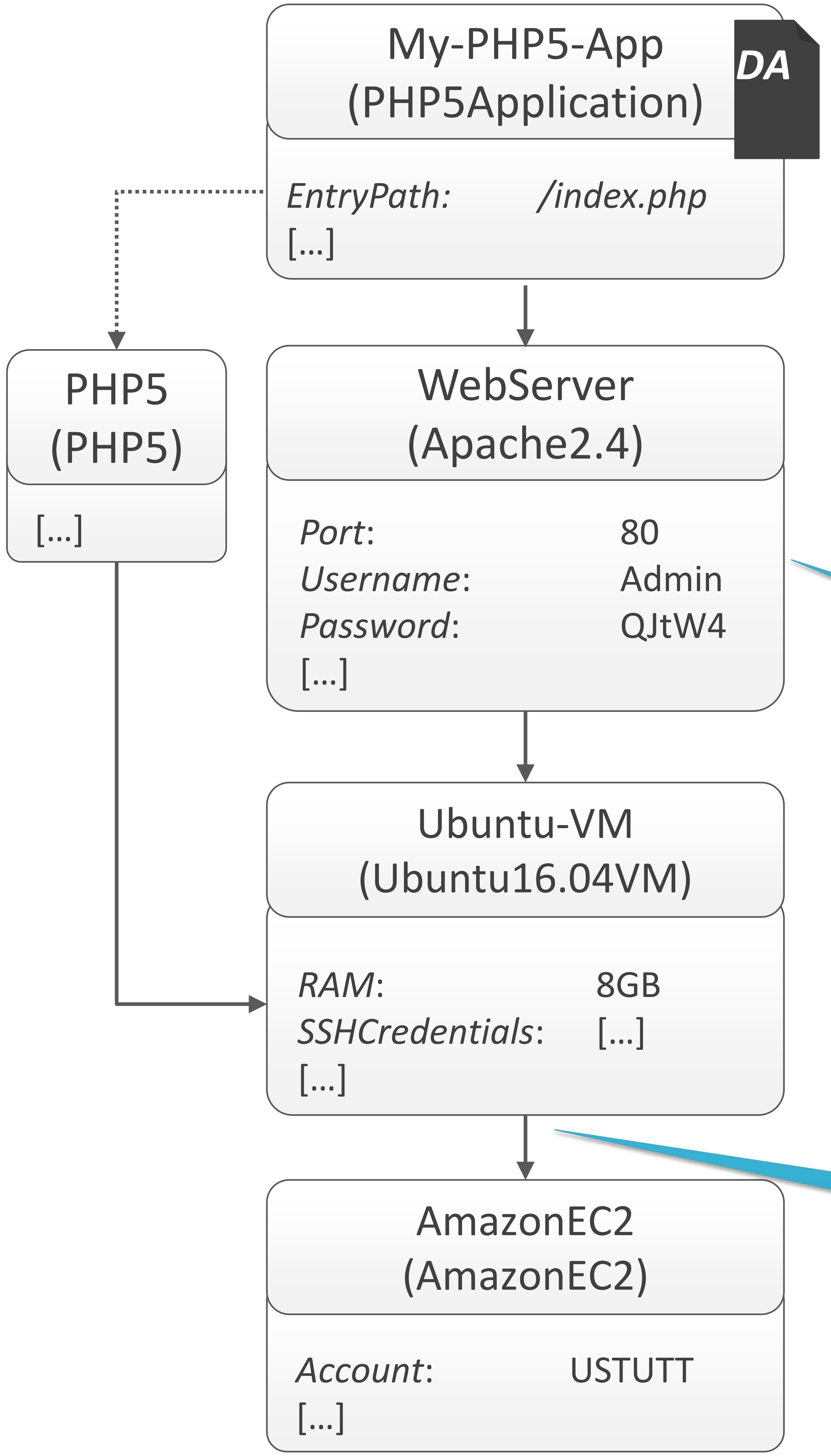


```
        - host:
            node: AwsPlatform
            relationship: con_HostedOn_1
            capability: host
        - invoker:
            node: AwsLambdaFunction
            relationship: con_AwsTriggers_0
            capability: invocable
    AwsLambdaFunction:
      type: radon.nodes.aws.AwsLambdaFunction
      metadata:
        x: "935"
        y: "242"
        displayName: "ThumbnailGenerator"
      properties:
        handler: "spblab.thumbgen.lambda.ThumbnailGenerationHandler"
        environment: "adasd"
        memory: 256
        name: "radon-particle-test"
        alias: "dev"
        runtime: "java8"
        statement_id: "lambda_test_permission01"
        zip_file: "thumbnail-generator-dev.jar"
```

Excerpt from a TOSCA model

- YAML-based specification
- Thumbnail generation deployed to AWS

Standards-based Deployment Modeling: TOSCA 101

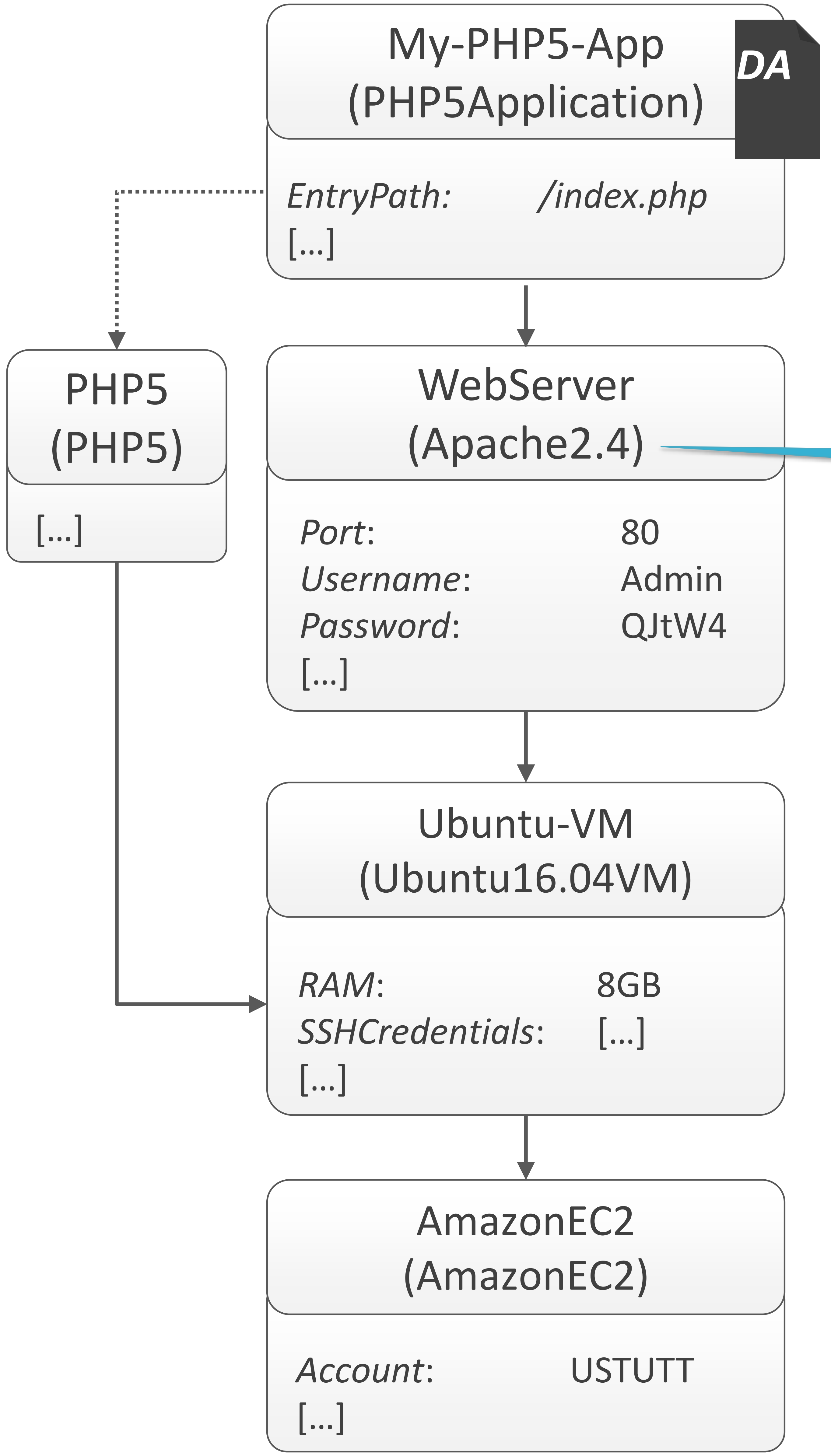


- TOSCA enables describing the structure of the application to be deployed in the form of a directed, acyclic graph
  - Nodes of the graph represent ***components***
    - e.g., an Apache Webserver, a VM, a PHP Application, or a MySQL database
    - These nodes are called ***Node Templates***
  - Edges of the graph represent ***relationships***
    - e.g., that one componenti is *hosted on* another component or *connects* to another component
    - These edges are called ***Relationship Templates***

Node Template

Relationship Template

Standards-based Deployment Modeling: TOSCA 101



- Both Node Templates and Relationship Templates are typed to define the semantics of templates
  - ***Node Types*** define the semantics of Node Templates
    - e.g., a Node Template may be of Node Type "Apache2.4"

Node Type

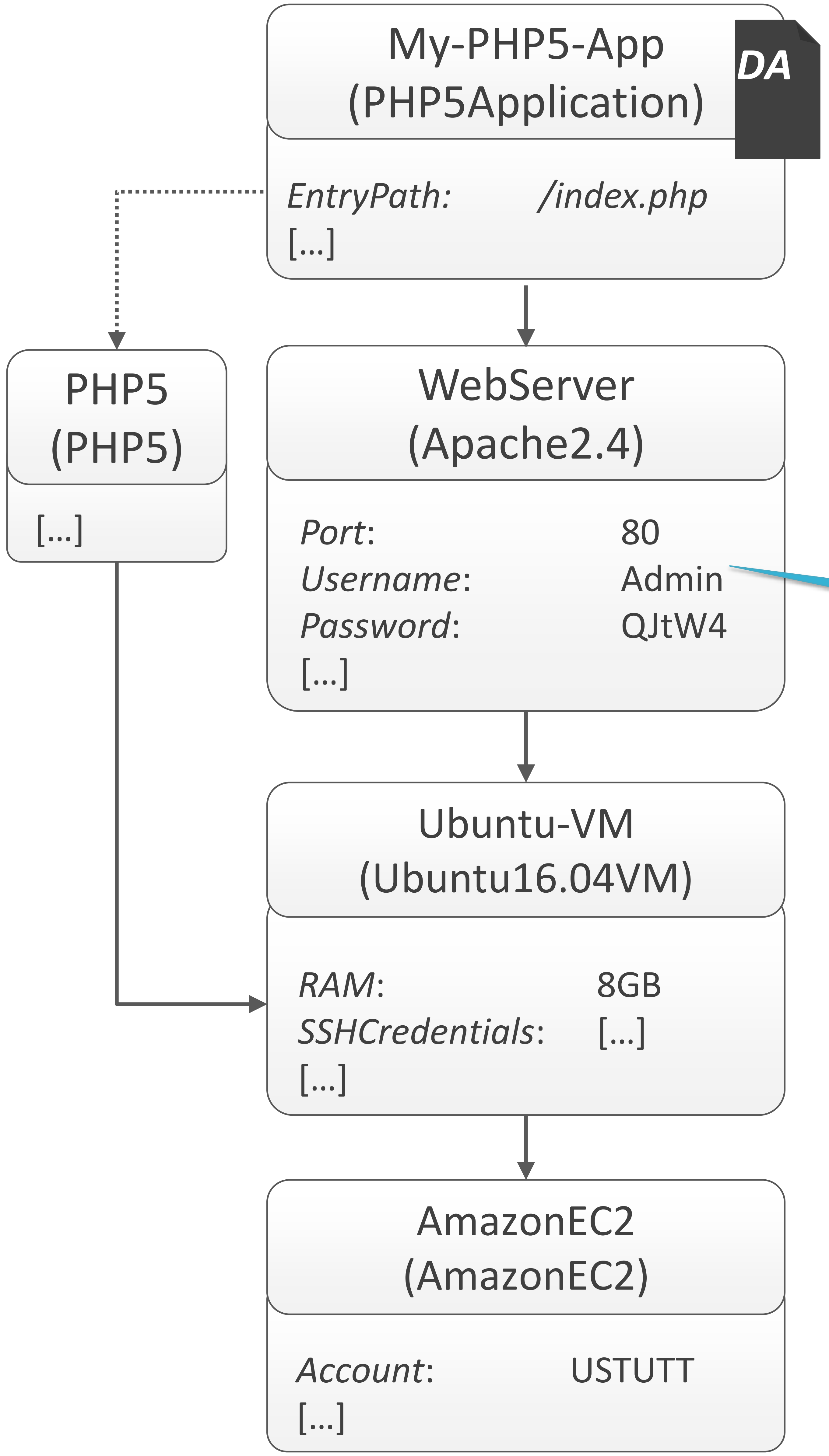
  - ***Relationship Types*** define the semantics of Relationship Templates
    - e.g., a Relationship Template may be of Relationship Type „hostedOn" or „SQLConnection"

→ = *hostedOn*

⇢ = *dependsOn*

  - The type system is extensible: New Node and Relationship Types can be defined arbitrarily
    - Also inheritance is supported

Standards-based Deployment Modeling: TOSCA 101

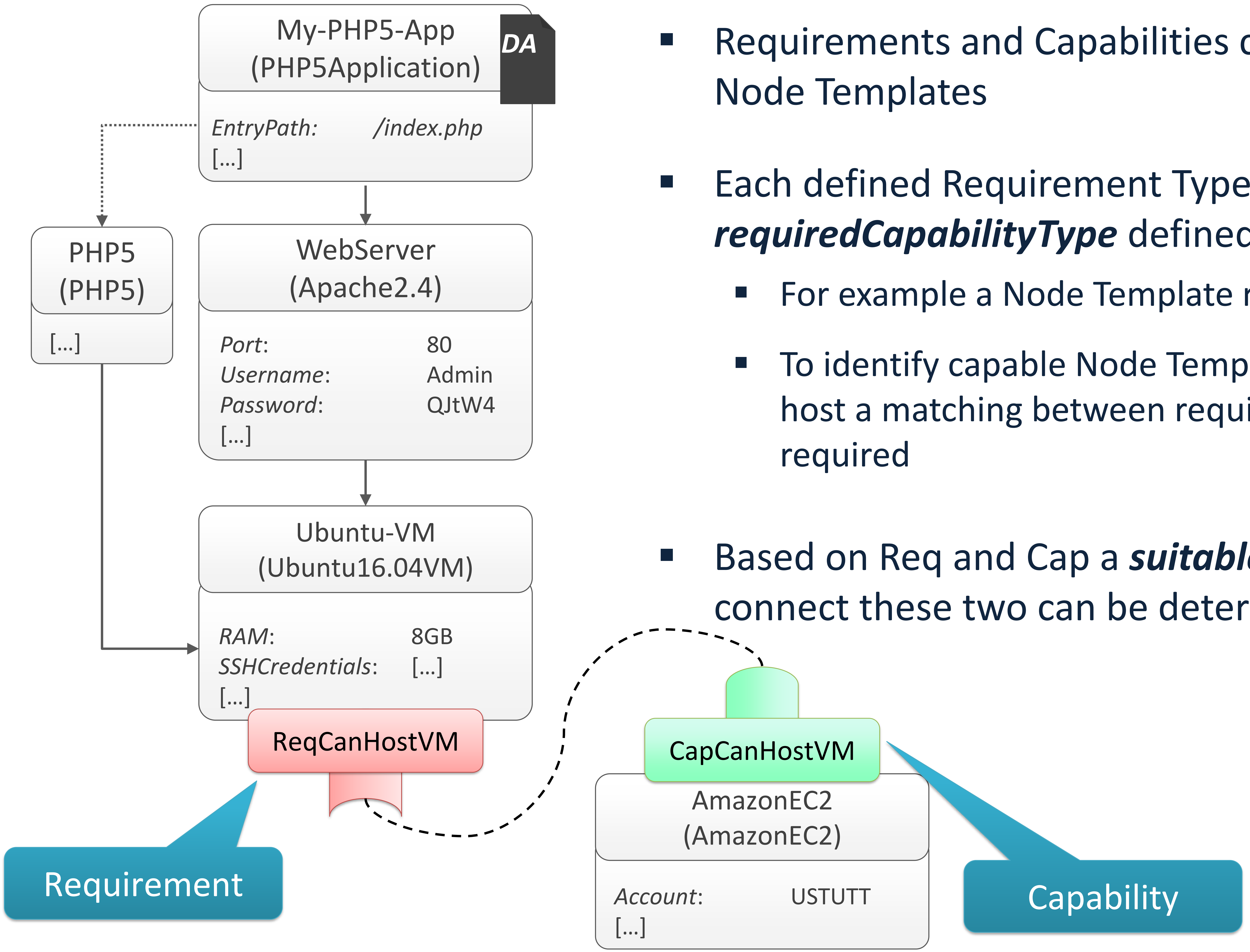


- To configure the deployment, Node and Relationship Templates may specify ***properties***
  - For example, to specify that the Apache Webserver shall serve HTTP requests at port 80
  - Or to specify the desired RAM of a virtual machine to be provisioned

Property

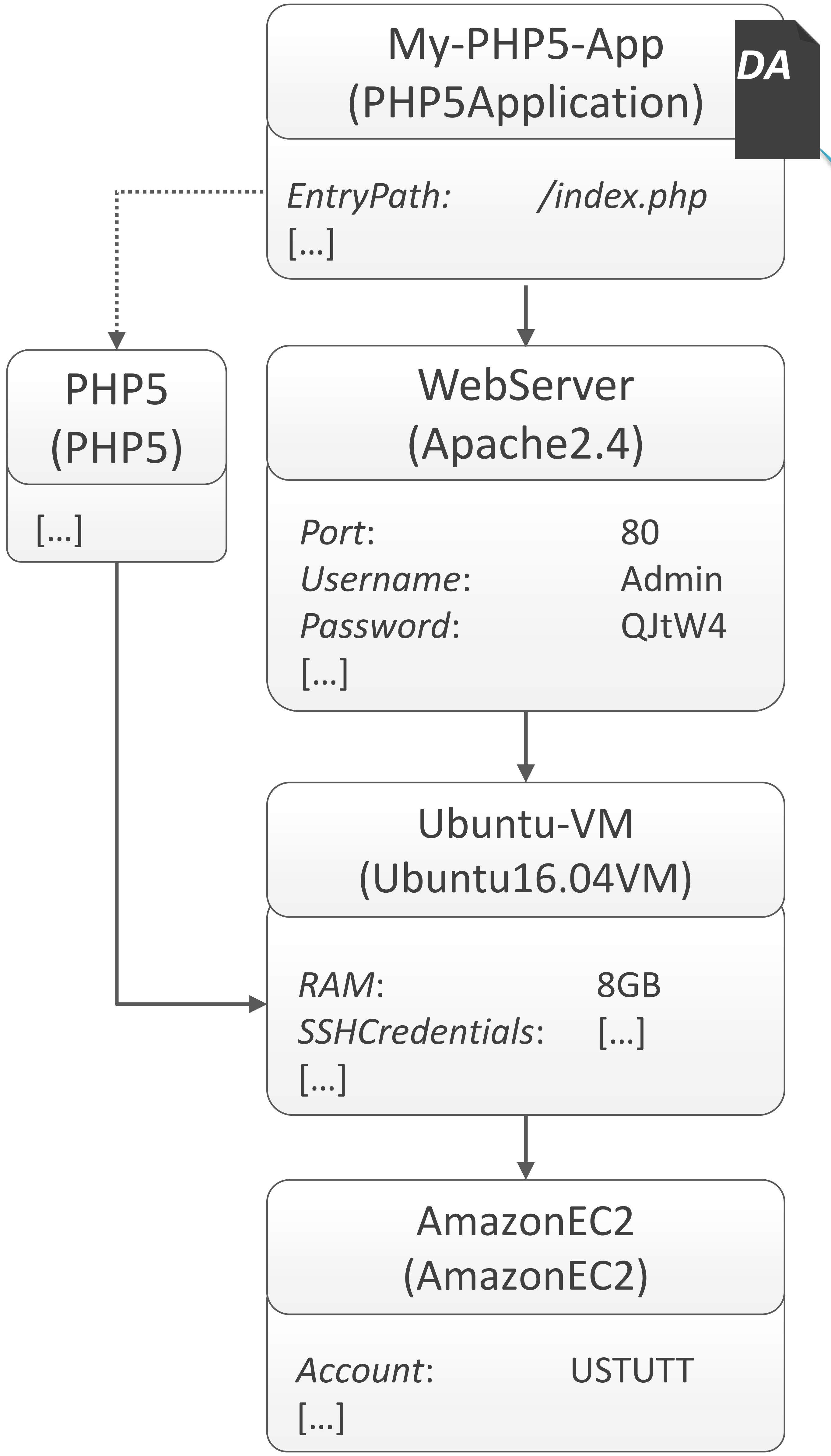
  - Properties may also contain ***instance information at runtime*** about a node or relationship
    - For example, the IP-address of a provisioned virtual machine, which is not known at modelling time
  - The properties a Node or Relationship Template provides and their schemas are defined by the respective Node or Relationship Type

Standards-based Deployment Modeling: TOSCA 101



- Requirements and Capabilities can be attached to Node Templates
- Each defined Requirement Types has a ***requiredCapabilityType*** defined (matchmaking)
  - For example a Node Template requires a host
  - To identify capable Node Templates able to serve as host a matching between requirement and capability is required
- Based on Req and Cap a ***suitable Relationship Type*** to connect these two can be determine

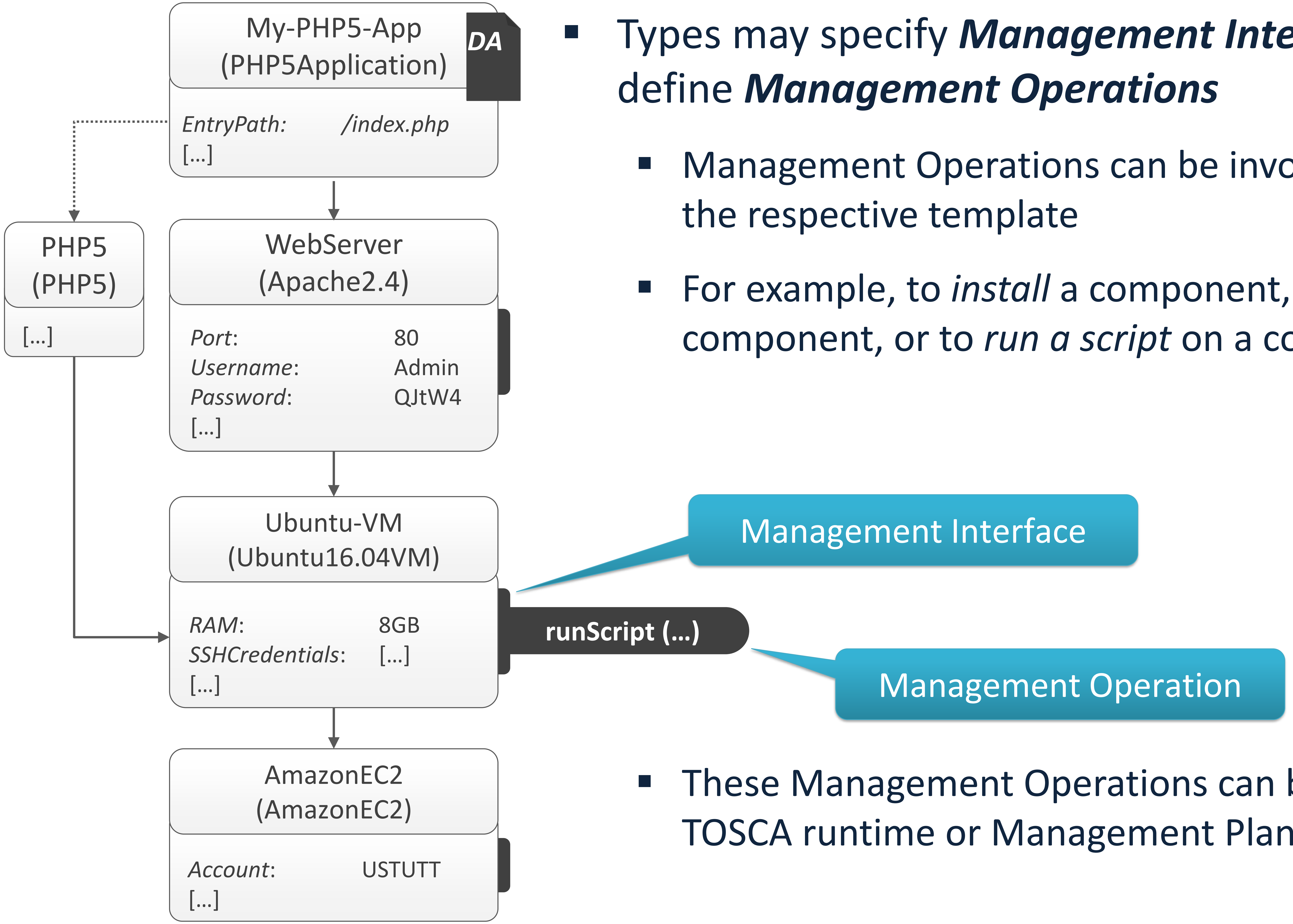
Standards-based Deployment Modeling: TOSCA 101



Deployment Artifact

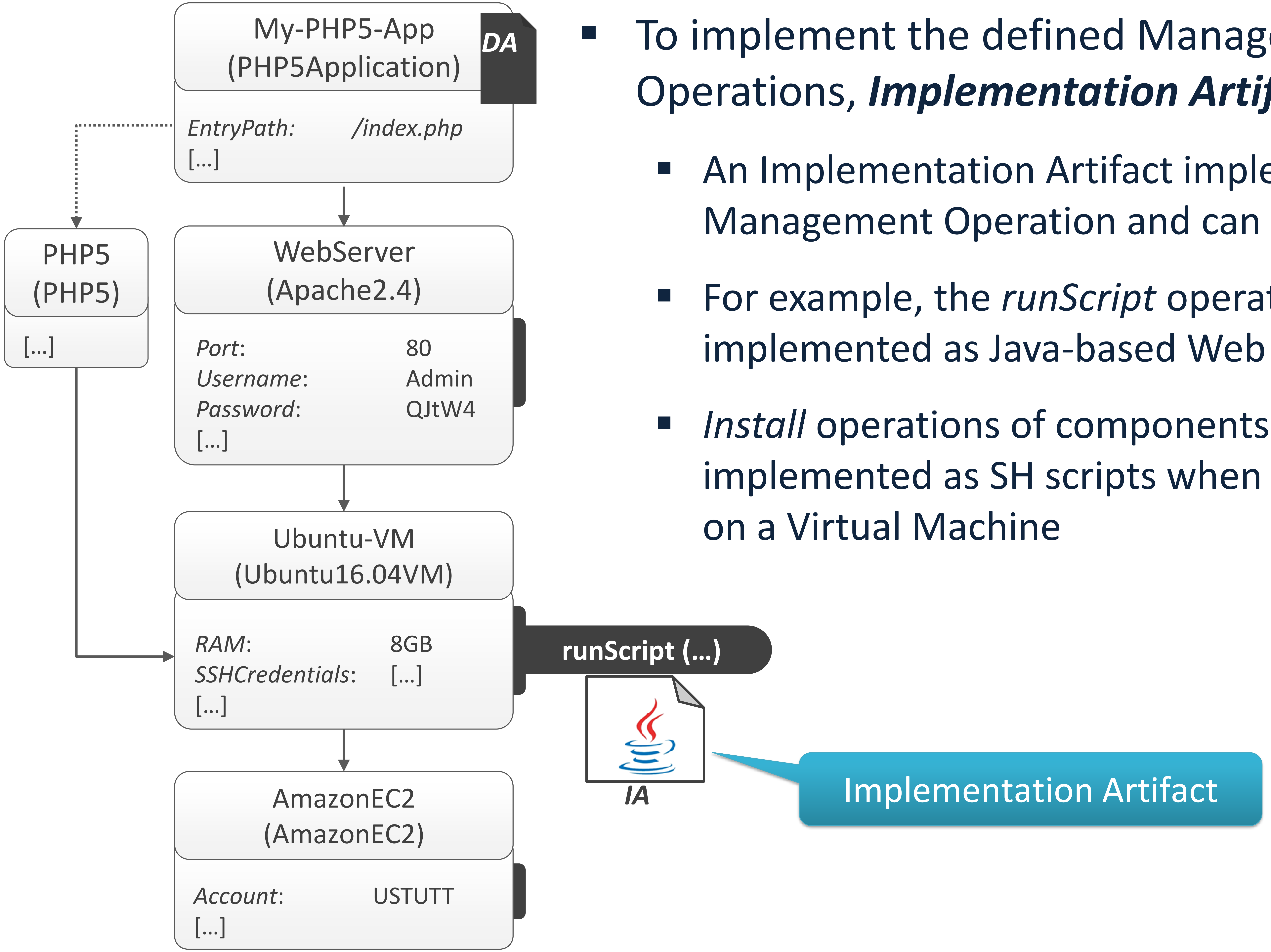
- To specify the implementations of components ***Deployment Artifacts (DA)*** are used
  - For example, a Deployment Artifact can be the PHP files of a Web application
  - A Deployment Artifact typically specifies one or more files and some properties about the artifact
    - For example, the type of the files

Standards-based Deployment Modeling: TOSCA 101



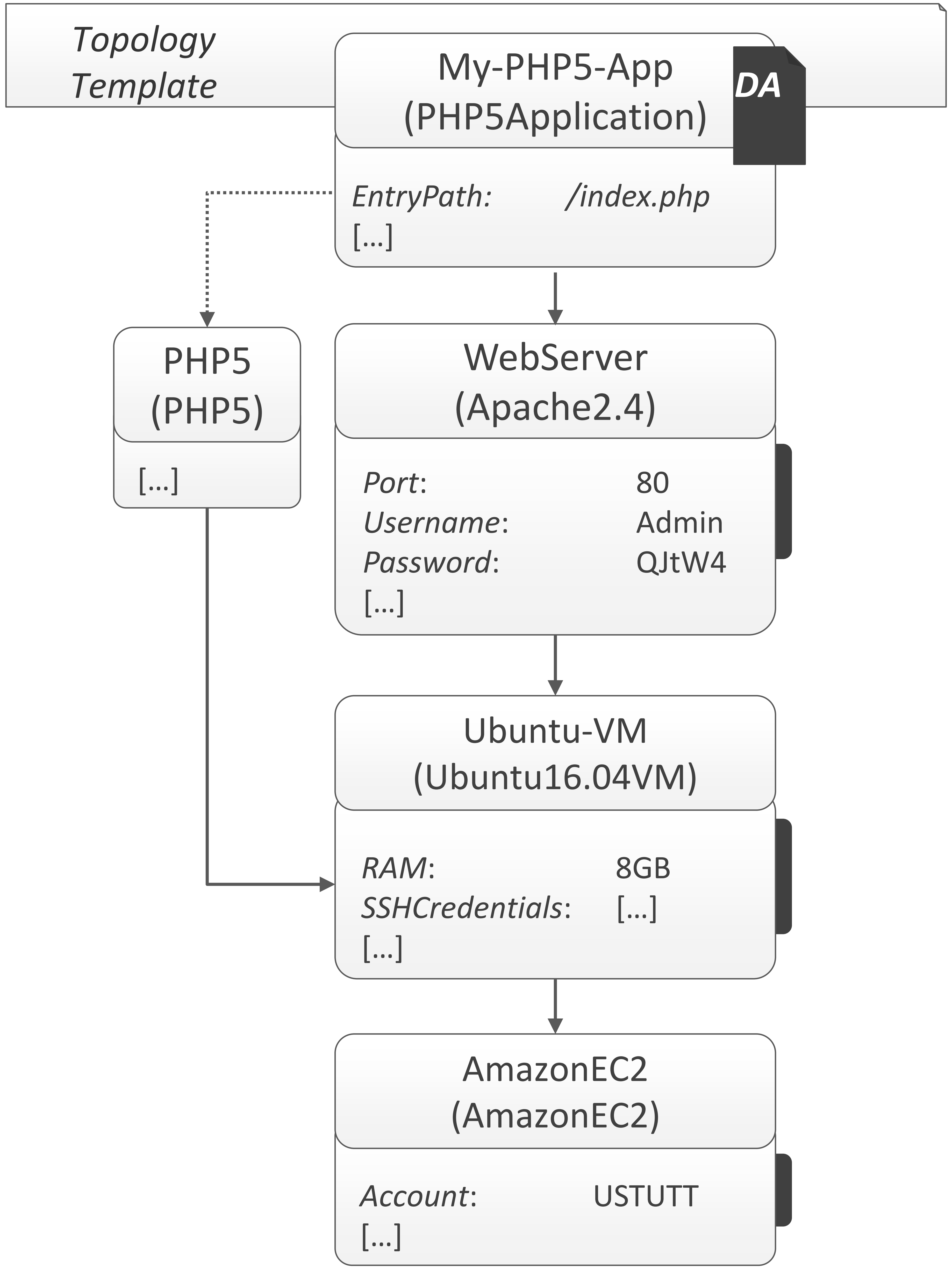
- Types may specify ***Management Interfaces*** that define ***Management Operations***
  - Management Operations can be invoked to manage the respective template
  - For example, to *install* a component, to *start* a component, or to *run a script* on a component
  - These Management Operations can be called by the TOSCA runtime or Management Plans (see next)

Standards-based Deployment Modeling: TOSCA 101



- To implement the defined Management Operations, ***Implementation Artifacts*** are used
  - An Implementation Artifact implements a certain Management Operation and can be executed
  - For example, the *runScript* operation could be implemented as Java-based Web Service
  - *Install* operations of components are often implemented as SH scripts when they shall be hosted on a Virtual Machine

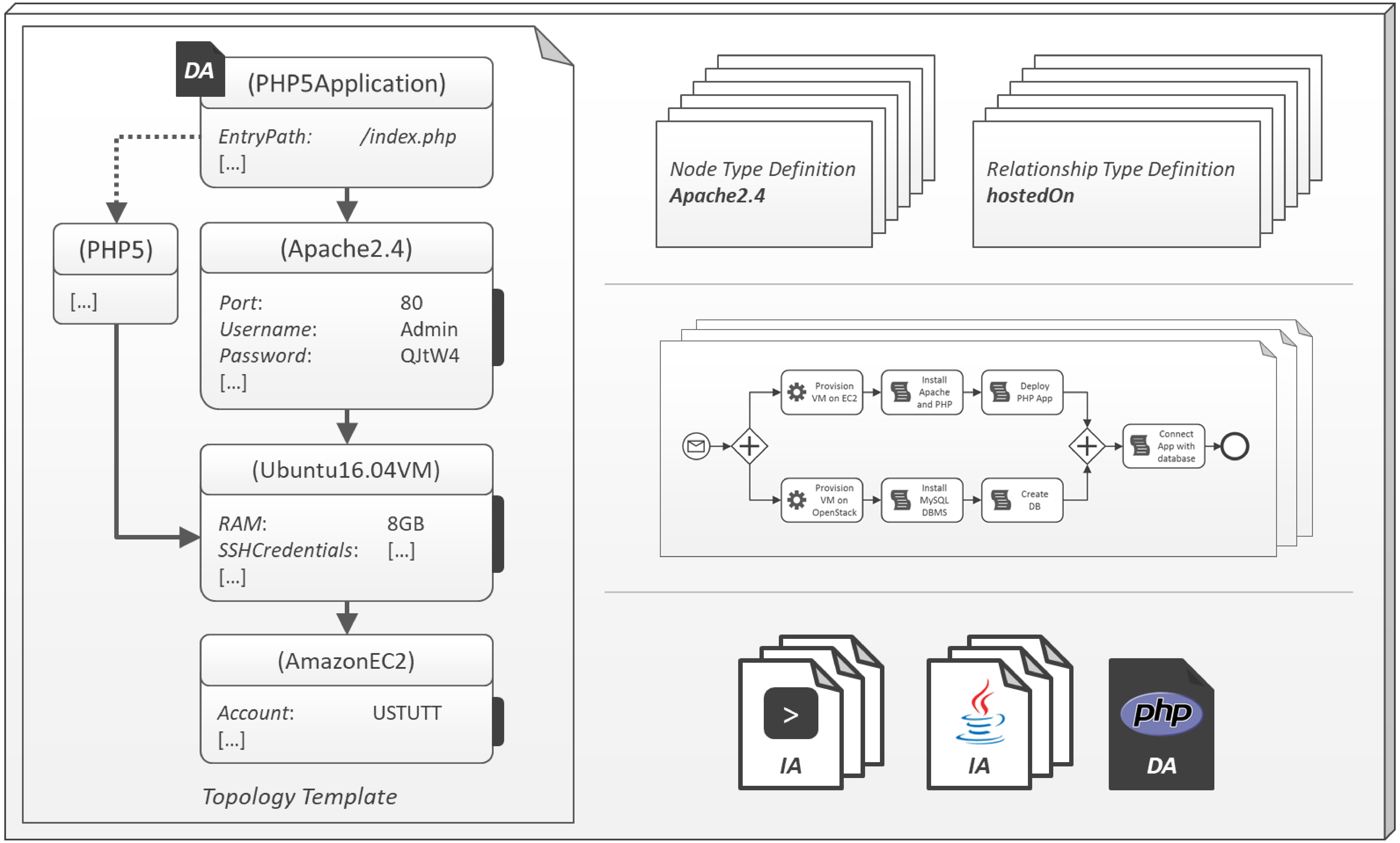
Standards-based Deployment Modeling: TOSCA 101



- A **Topology Template** represents the deployment model with all Node and Relationship Templates of the application
- A **Service Template** contains one or more Topology Templates as well as all used type definitions and artifacts
  - A Service Template can be used also to package only type definitions or artifacts
- A **Cloud Service Archive (CSAR)** is an archive format standardized by TOSCA to package Service Templates as well as all required files, plans, etc. into a ZIP file

Standards-based Deployment Modeling: TOSCA 101

Cloud Service Archive (CSAR)



DA (PHP5Application)

EntryPath: */index.php*
[…]

(PHP5)
[…]

(Apache2.4)

Port: 80
Username: Admin
Password: QJtW4
[…]

(Ubuntu16.04VM)

RAM: 8GB
SSHCredentials: […]
[…]

(AmazonEC2)

Account: USTUTT
[…]

Topology Template

Node Type Definition
Apache2.4

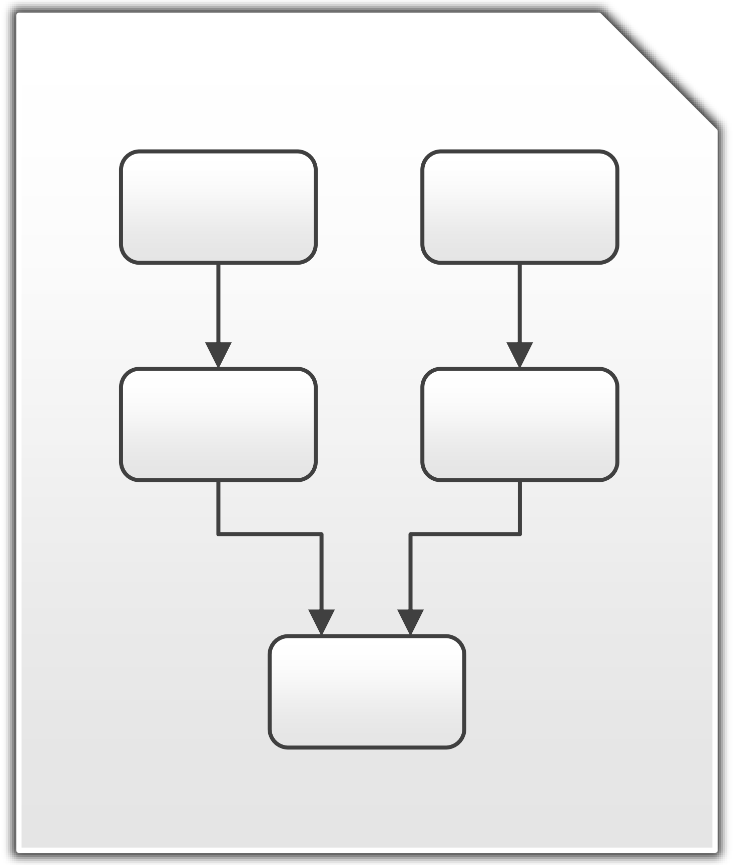
Relationship Type Definition
hostedOn

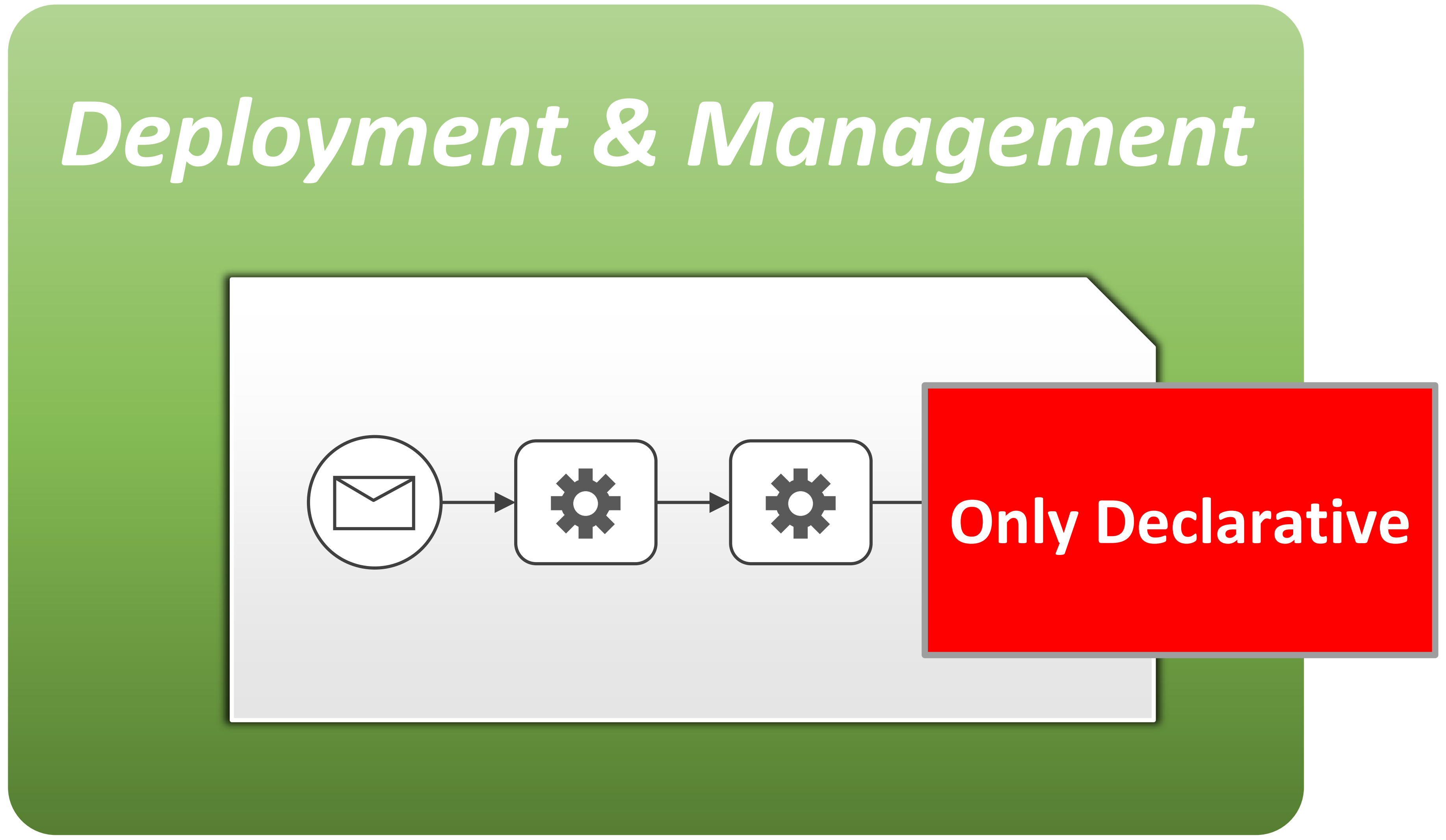
Standards-based Deployment Modeling: TOSCA 101

TOSCA:

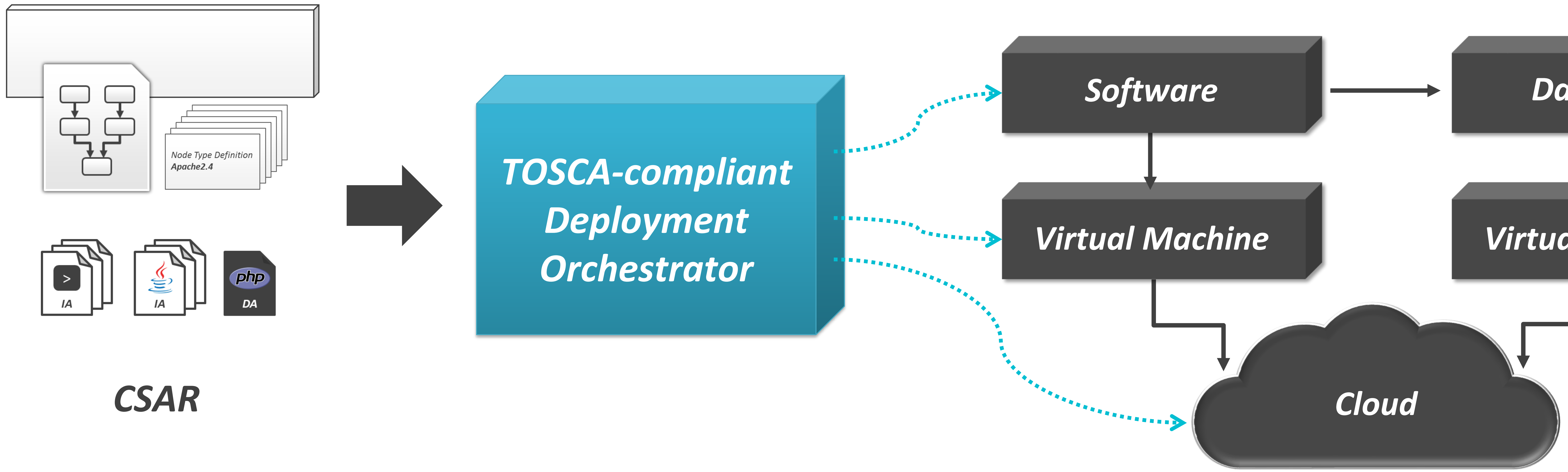
Topology and **Orchestration**
Specification for Cloud Applications

Application Structure





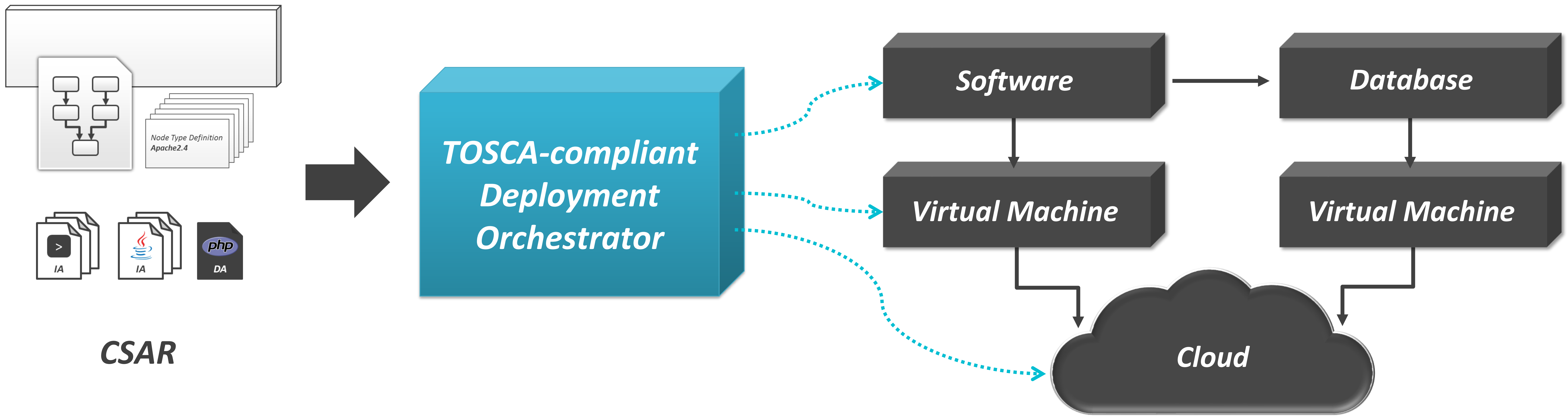
Standards-based Deployment Modeling: TOSCA 101



A TOSCA-compliant Orchestrator

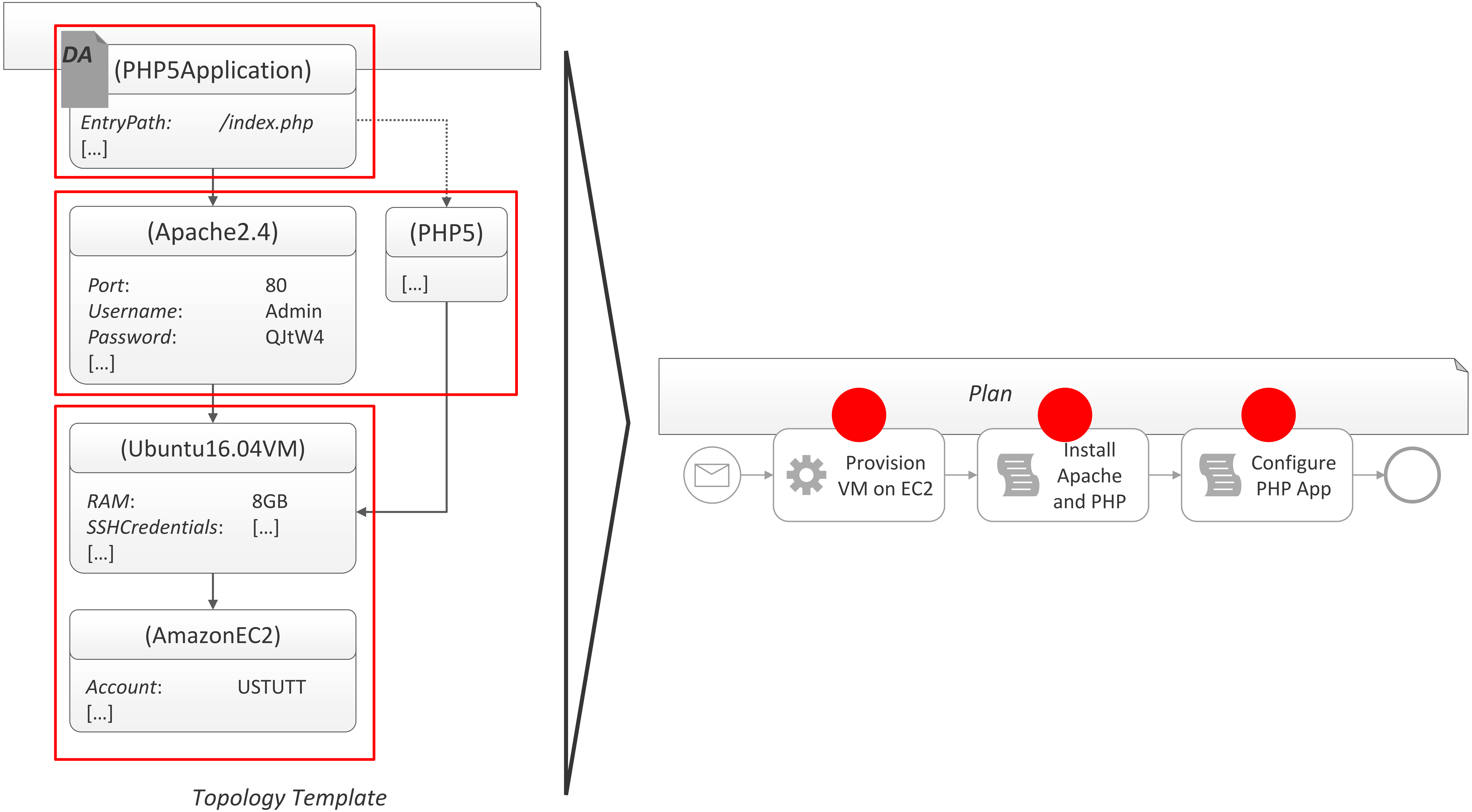
- provisions modelled applications using the Topology Template and provided IAs
- Hence, TOSCA enables the declarative deployment modelling

Standards-based Deployment Modeling: TOSCA 101



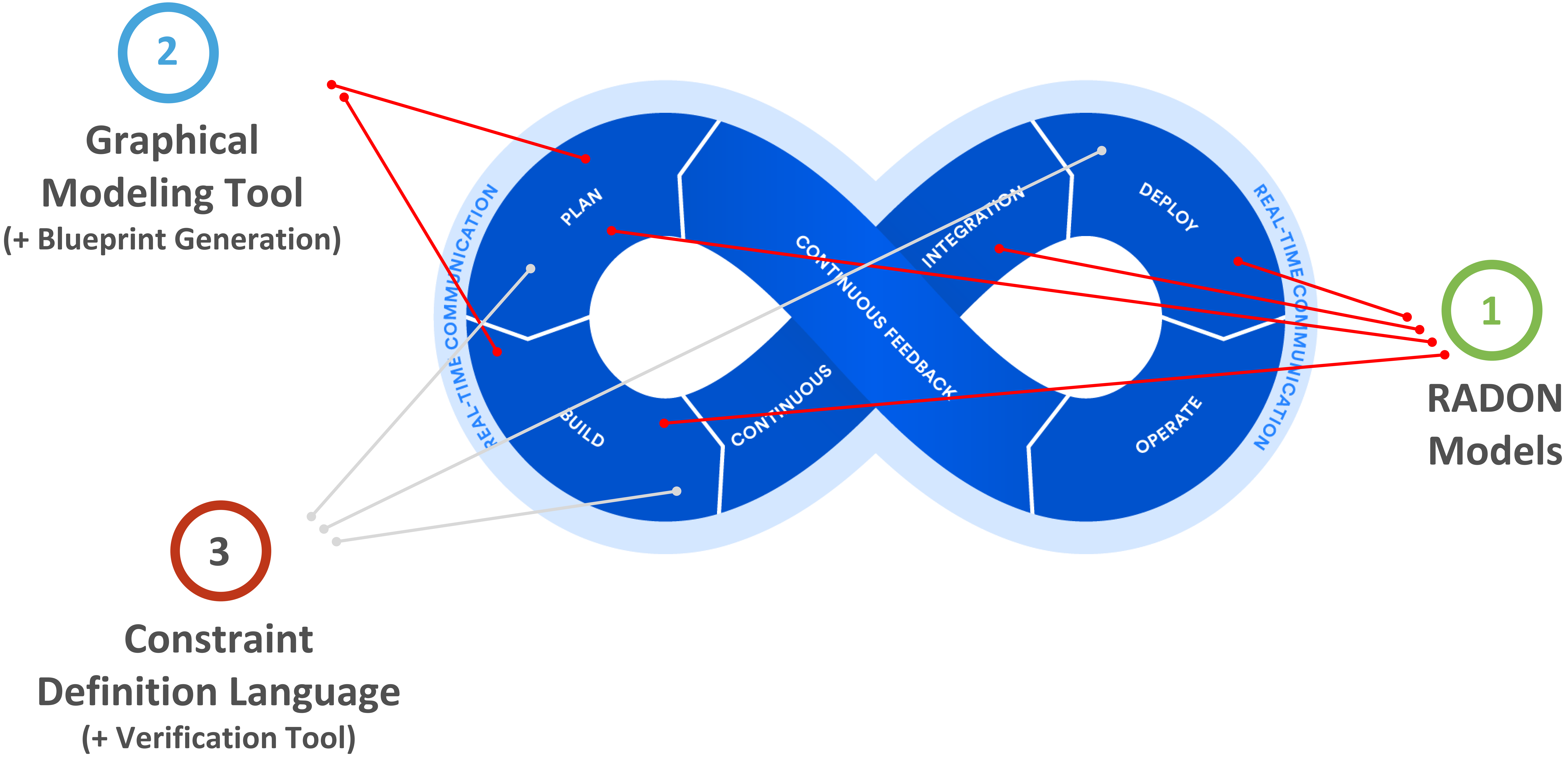
A declarative runtime interprets the model based on defined semantics

- Lifecycle Interface operations, e.g., **create, start, configure, stop, delete**
- HostedOn relationships enable deriving the provisioning order of components and executes the required Lifecycle operations install, start, and configure in this order for each component


DA
(PHP5Application)
EntryPath: /index.php
[...]
(Apache2.4)
Port: 80
Username: Admin
Password: QJtW4
[...]
(PHP5)
[...]
(Ubuntu16.04VM)
RAM: 8GB
SSHCredentials: [...]
[...]
(AmazonEC2)
Account: USTUTT
[...]
Topology Template
Plan
Provision VM on EC2
Install Apache and PHP
Configure PHP App

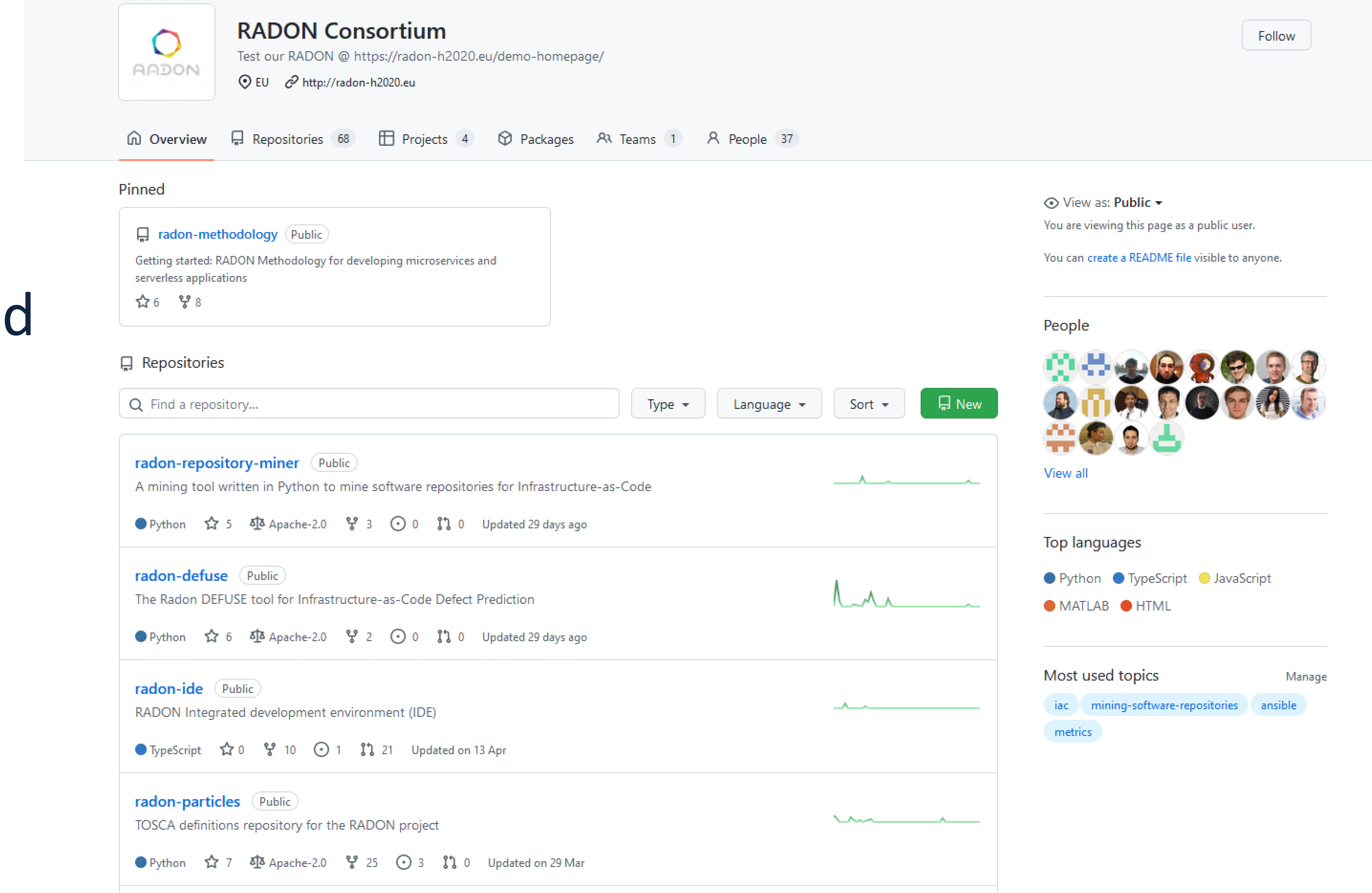
RADON Framework

Modeling in RADON



RADON Models

- TOSCA-based modeling profile
- Modeling of abstract and platform-specific serverless functions
- Function types: invocable/scheduled
- Event specification using TOSCA relationships, data types based on CloudEvents spec (https://cloudevents.io/)
- A set of data pipeline types



RADON Consortium

Test our RADON @ https://radon-h2020.eu/demo-homepage/

EU http://radon-h2020.eu

Pinned

radon-methodology Public

Getting started: RADON Methodology for developing microservices and serverless applications

☆ 6 8

Repositories

radon-repository-miner Public

A mining tool written in Python to mine software repositories for Infrastructure-as-Code

● Python ☆ 5 Apache-2.0 3 0 0 Updated 29 days ago

radon-defuse Public

The Radon DEFUSE tool for Infrastructure-as-Code Defect Prediction

● Python ☆ 6 Apache-2.0 2 0 0 Updated 29 days ago

radon-ide Public

RADON Integrated development environment (IDE)

● TypeScript ☆ 0 10 1 21 Updated on 13 Apr

radon-particles Public

TOSCA definitions repository for the RADON project

● Python ☆ 7 Apache-2.0 25 3 0 Updated on 29 Mar

View as: **Public**

You are viewing this page as a public user.

You can create a README file visible to anyone.

People

Top languages

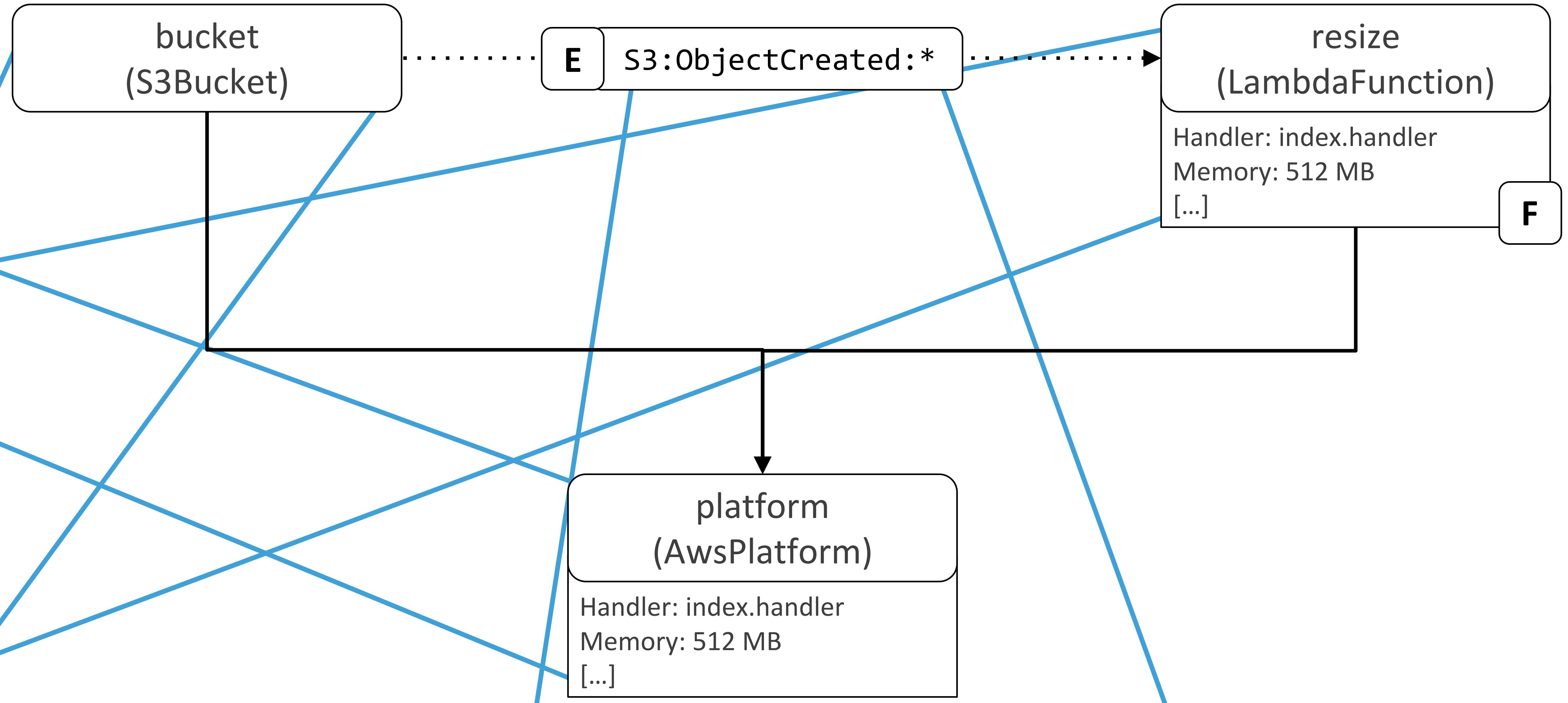
● Python ● TypeScript ● JavaScript ● MATLAB ● HTML

Most used topics

iac mining-software-repositories ansible metrics

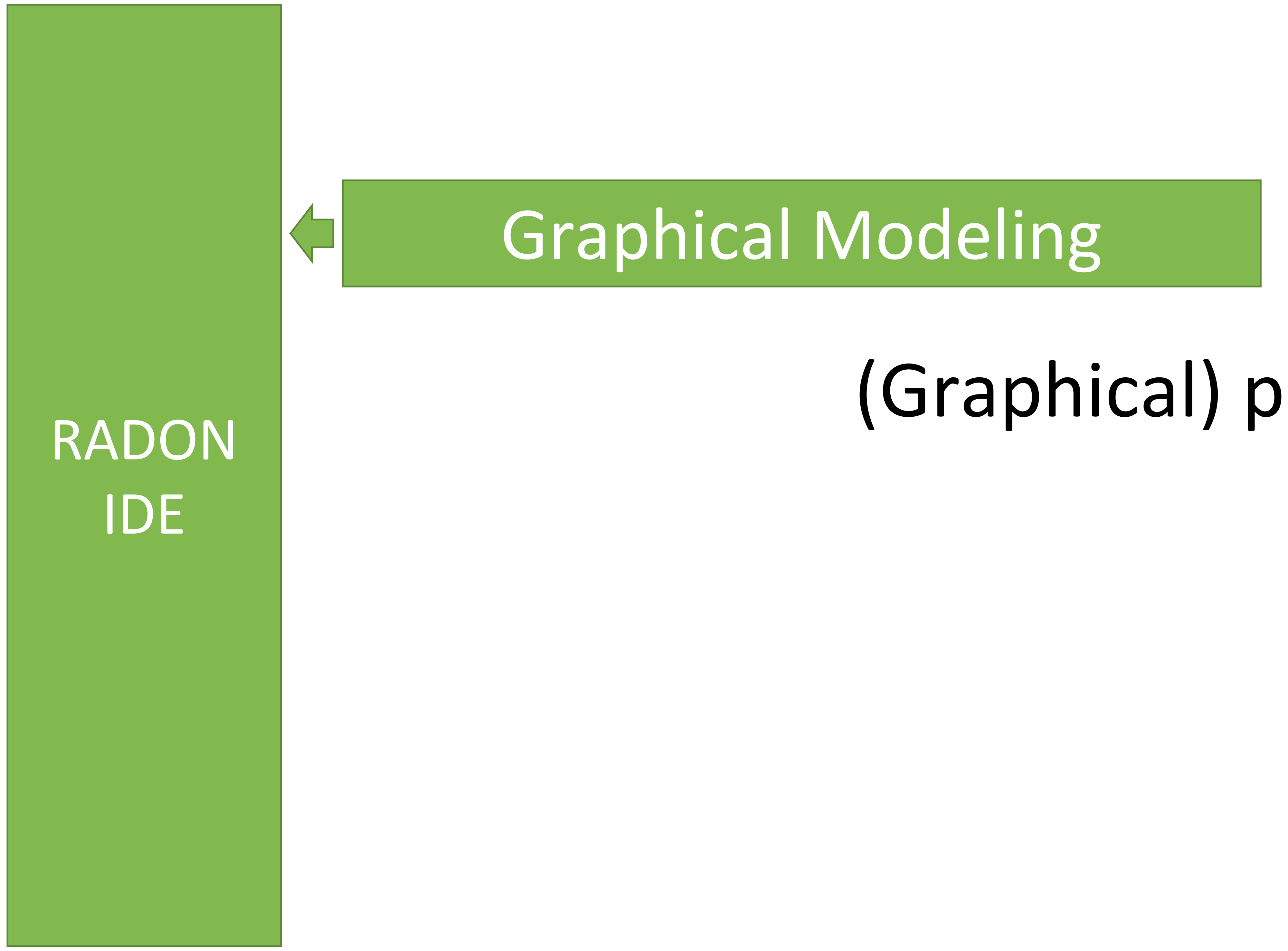
RADON Models

```
tosca_definitions_version: tosca_simple_yaml_1_3
 topology_template:
  node_templates:
   platform:
    type: radon.nodes.aws.AwsPlatform
    properties:
       # omitted for brevity
   resize:
    type: radon.nodes.aws.LambdaFunction
    properties:
     handler: index.handler
     memory: 512
       # ...
    artifacts:
     deployment_package:
      file: thumbnail.zip
      type: radon.artifacts.archive.Zip
    requirements:
     - host: platform
   bucket:
    type: radon.nodes.aws.S3Bucket
    requirements:
     - host: platform
     - invoker:
        node: resize
        relationship: trigger
```



```
relationship_templates:
  trigger:
   type: radon.relationships.aws.Triggers
   properties:
    event_types:
     - wildcard:
         type: radon.datatypes.Event
         properties:
          type: s3:ObjectCreated:*
```

RADON Framework Overview

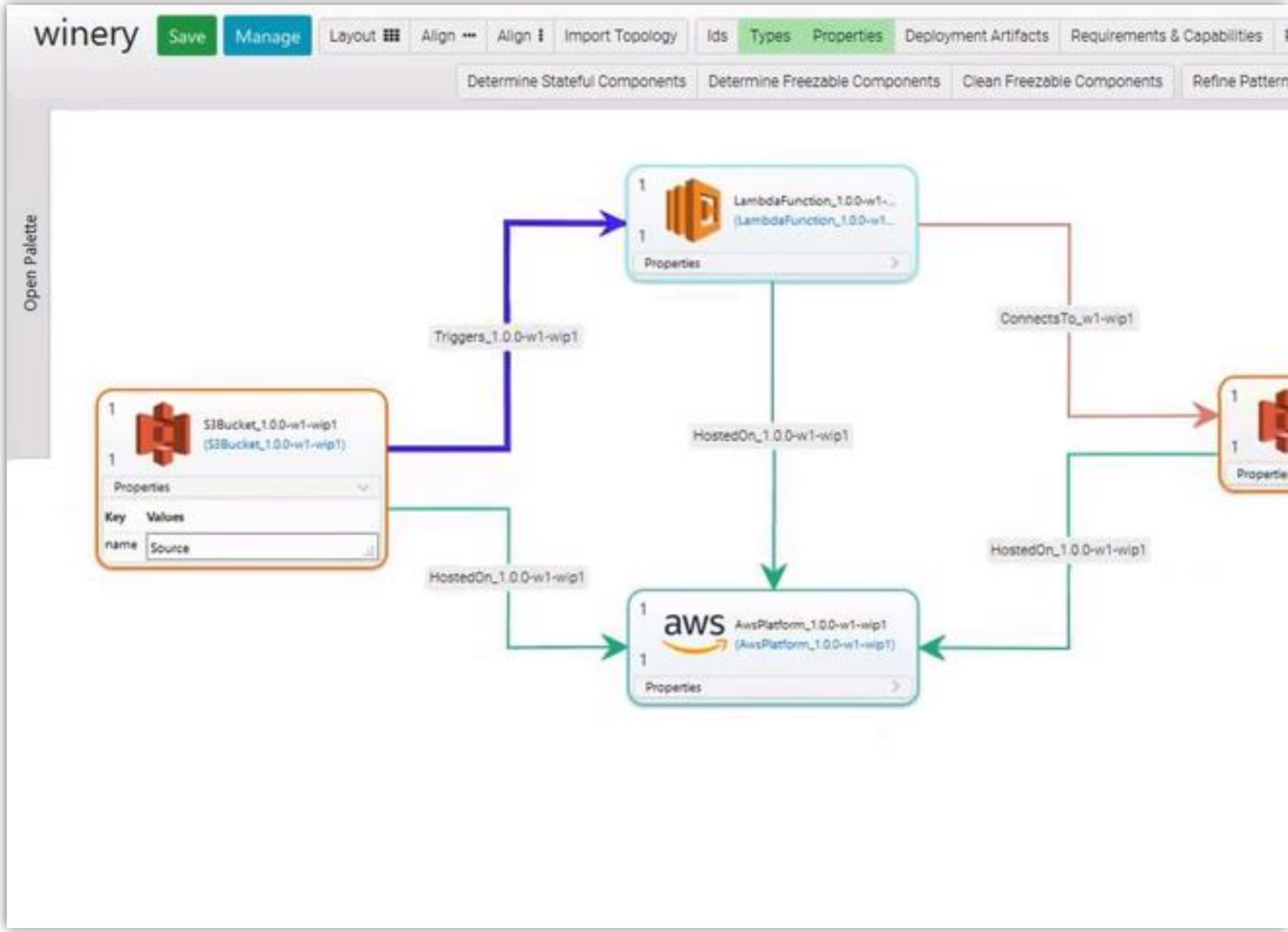


(Graphical) provider-agnostic application modeling

Graphical Modeling Tool

- Extended Eclipse Winery as modeling environment
- Official eclipse project

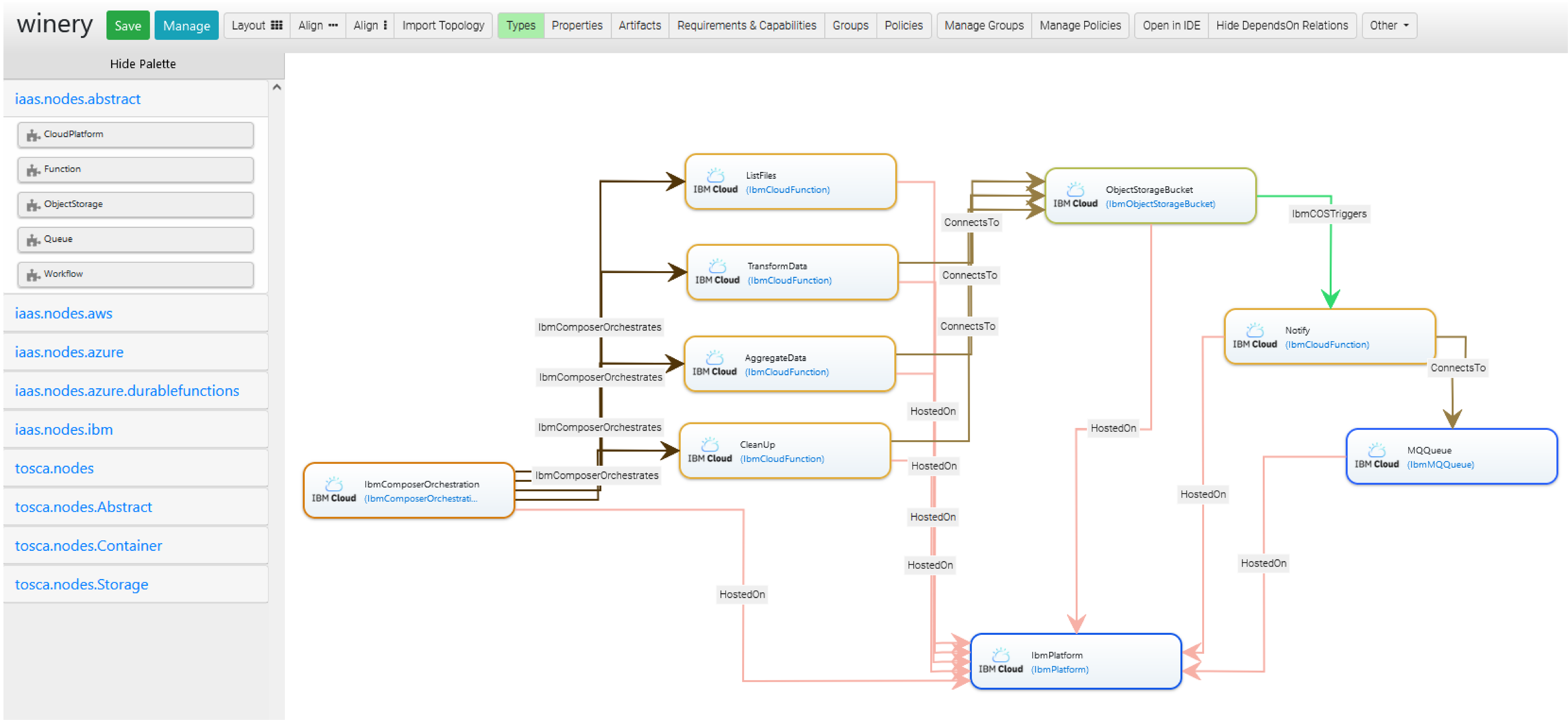
- Web-based environment
- Manage TOSCA types, templates and related artifacts
- Graphically model TOSCA topologies
- Added support for TOSCA YAML 1.3



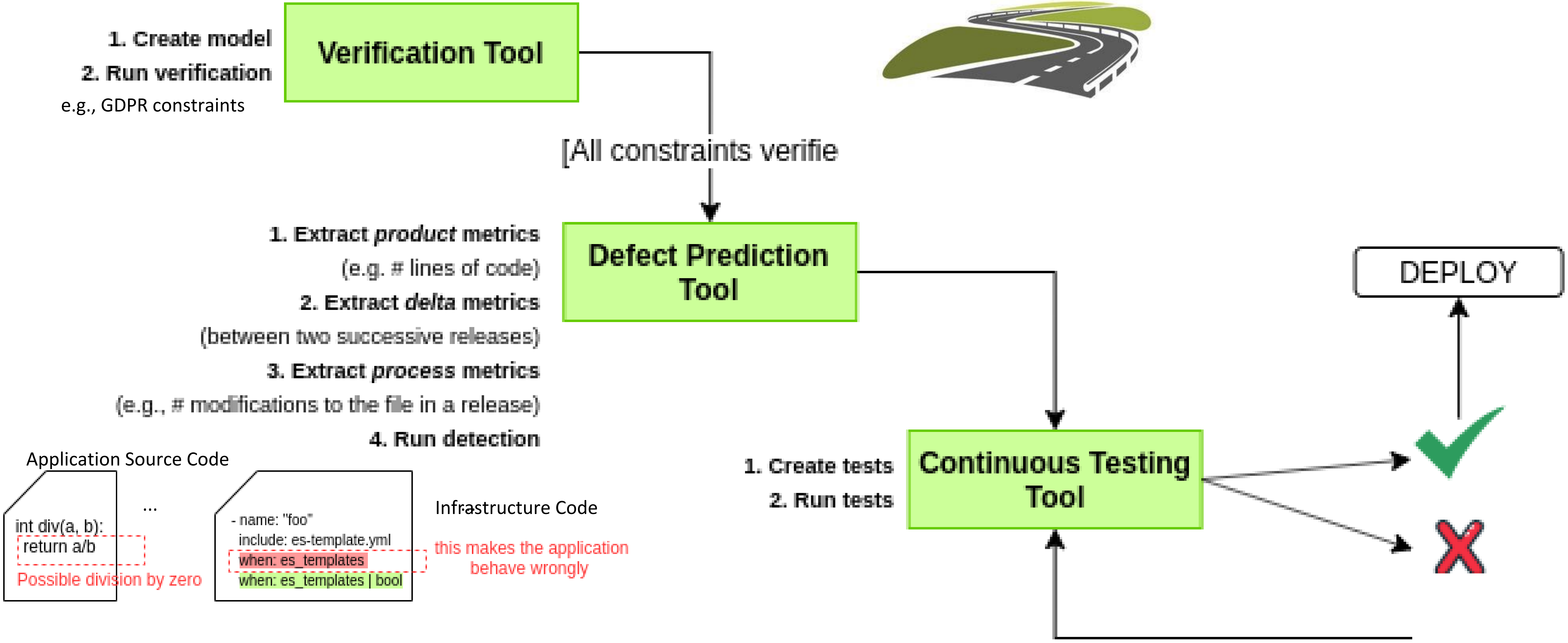
Modelling FaaS-based Applications in RADON

aws apptokens (AwsDynamoDBTable)
aws organizations (AwsDynamoDBTable)
aws AWS_platform_US (AwsPlatform)
aws browser (AwsLambdaFunction)
aws user_logout (AwsLambdaFunction)
aws create_app_token (AwsLambdaFunction)
aws create_org (AwsLambdaFunction)
aws list_organization (AwsLambdaFunction)
aws list_repo_public (AwsLambdaFunction)
aws list_artifact (AwsLambdaFunction)
aws artifact_details (AwsLambdaFunction)
aws list_version (AwsLambdaFunction)
aws create_repo (AwsLambdaFunction)
aws list_repo_private (AwsLambdaFunction)
aws create_user (AwsLambdaFunction)
aws user_login (AwsLambdaFunction)
aws user_info (AwsLambdaFunction)
aws upload_artifact (AwsLambdaFunction)
aws artifact_download (AwsLambdaFunction)
aws users (AwsDynamoDBTable)
aws artifacts-dev (AwsDynamoDBTable)
aws repositories (AwsDynamoDBTable)
aws AWS_platform_EU (AwsPlatform)
aws AwsApiGateway (AwsApiGateway)
aws radon-repository-testing (AwsS3Bucket)
ConnectsTo
HostedOn
ApiGatewayTriggers

Modelling FaaS-based Applications in RADON



Quality Guardrails in RADON



1. Create model
2. Run verification

e.g., GDPR constraints

1. Extract *product* metrics (e.g. # lines of code)
2. Extract *delta* metrics (between two successive releases)
3. Extract *process* metrics (e.g., # modifications to the file in a release)
4. Run detection

Application Source Code

```
int div(a, b):
 return a/b
```

Possible division by zero

...

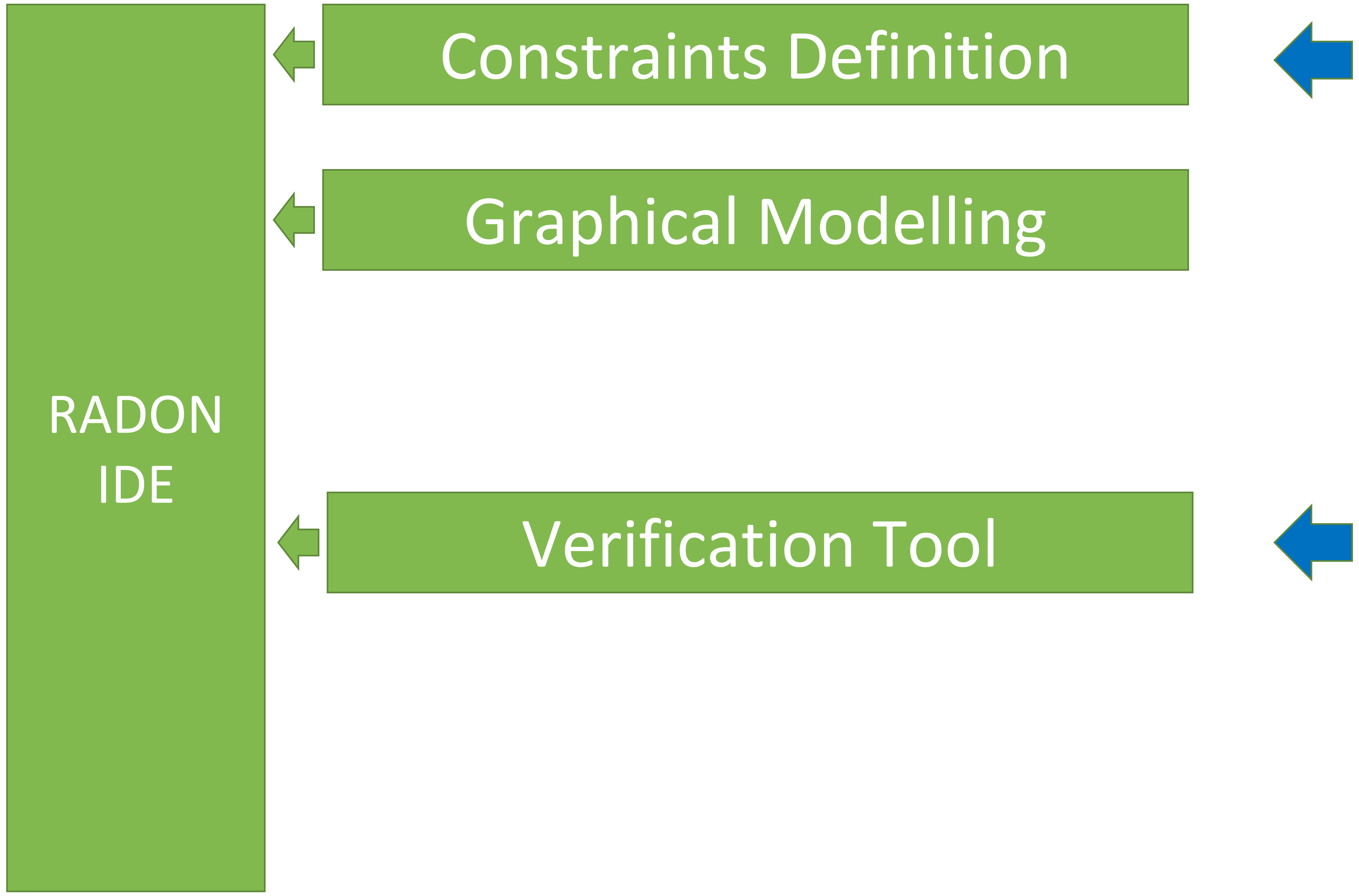
Infrastructure Code

```
- name: "foo"
  include: es-template.yml
  when: es_templates
  when: es_templates | bool
```

this makes the application behave wrongly

1. Create tests
2. Run tests

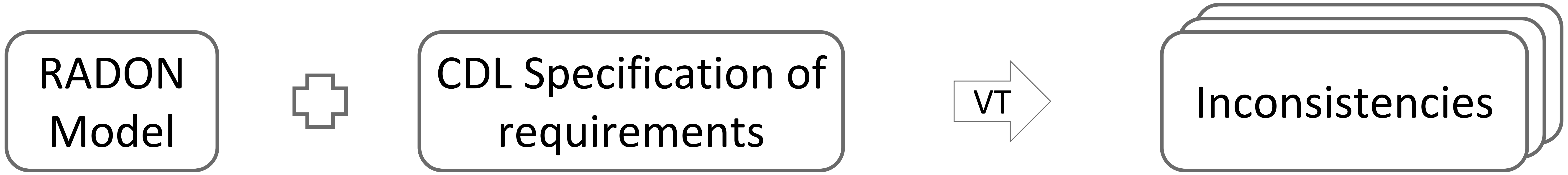
RADON Framework Overview



Define application model constraints to be verified

Verification of expensive constraints (e.g., privacy, security, design pattern violations) before deployment

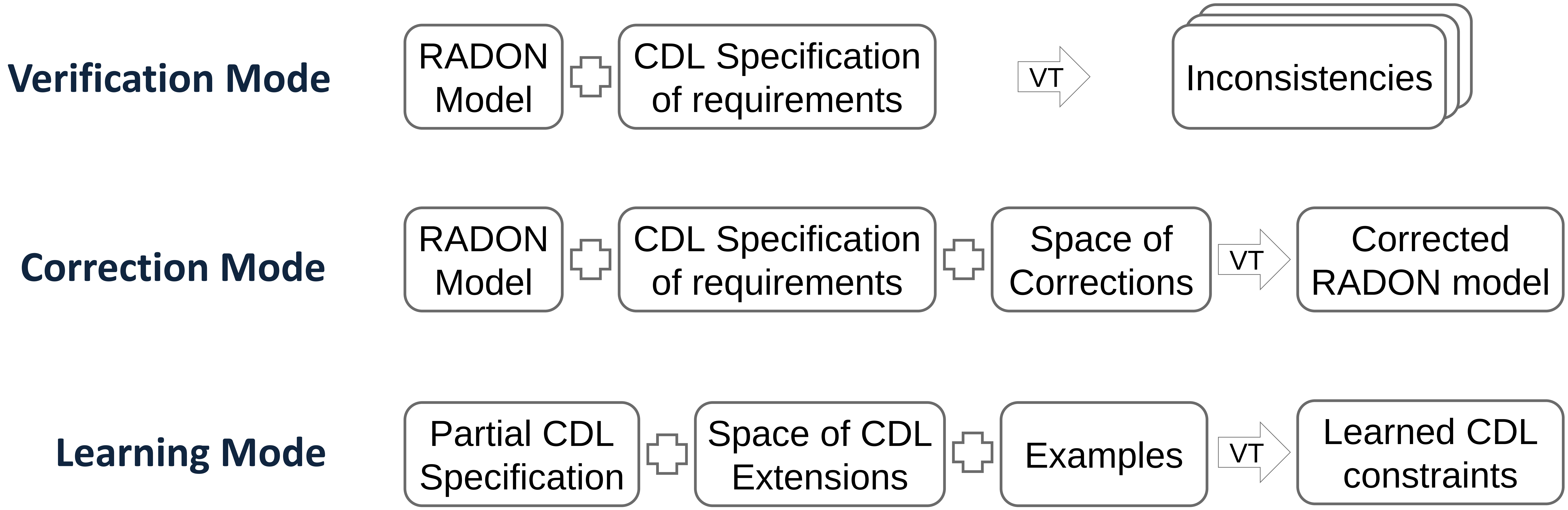
Constraint Definition Language



- Express functional and non-functional requirements on a RADON model
- Built-in definitions of common runtime issues, such as deadlocks, race conditions and execution loops + custom, user-defined definitions
- Verification Tool verifies that RADON models meet specifications in the CDL
- VT can be used at design-time, to search for issues that could occur at run-time

Verification Tool Modes

Verification Tool supports the following modes:



Verification Mode

RADON Model + CDL Specification of requirements VT → Inconsistencies

Correction Mode

RADON Model + CDL Specification of requirements + Space of Corrections VT → Corrected RADON model

Learning Mode

Partial CDL Specification + Space of CDL Extensions + Examples VT → Learned CDL constraints

Verification Example

aws
uploads_1
(AwsS3Bucket)
Properties
AwsTriggers
HostedOn
aws
uploads_2
(AwsS3Bucket)
Properties
AwsTriggers
HostedOn
aws
create_thumbnail
(AwsLambdaFunction)
Properties
ConnectsTo
Database
(Database)
Properties
Input
Input
VT

Key	Values
role_name	Edit your value here.
schedule	~
handler	index.handler
environment	prod
memory	512
name	resize
runtime	nodejs
timeout	3

secret_access_key	{ get_input:
region	eu-west-2

secret_access_key	{ get_input:
region	eu-west-1

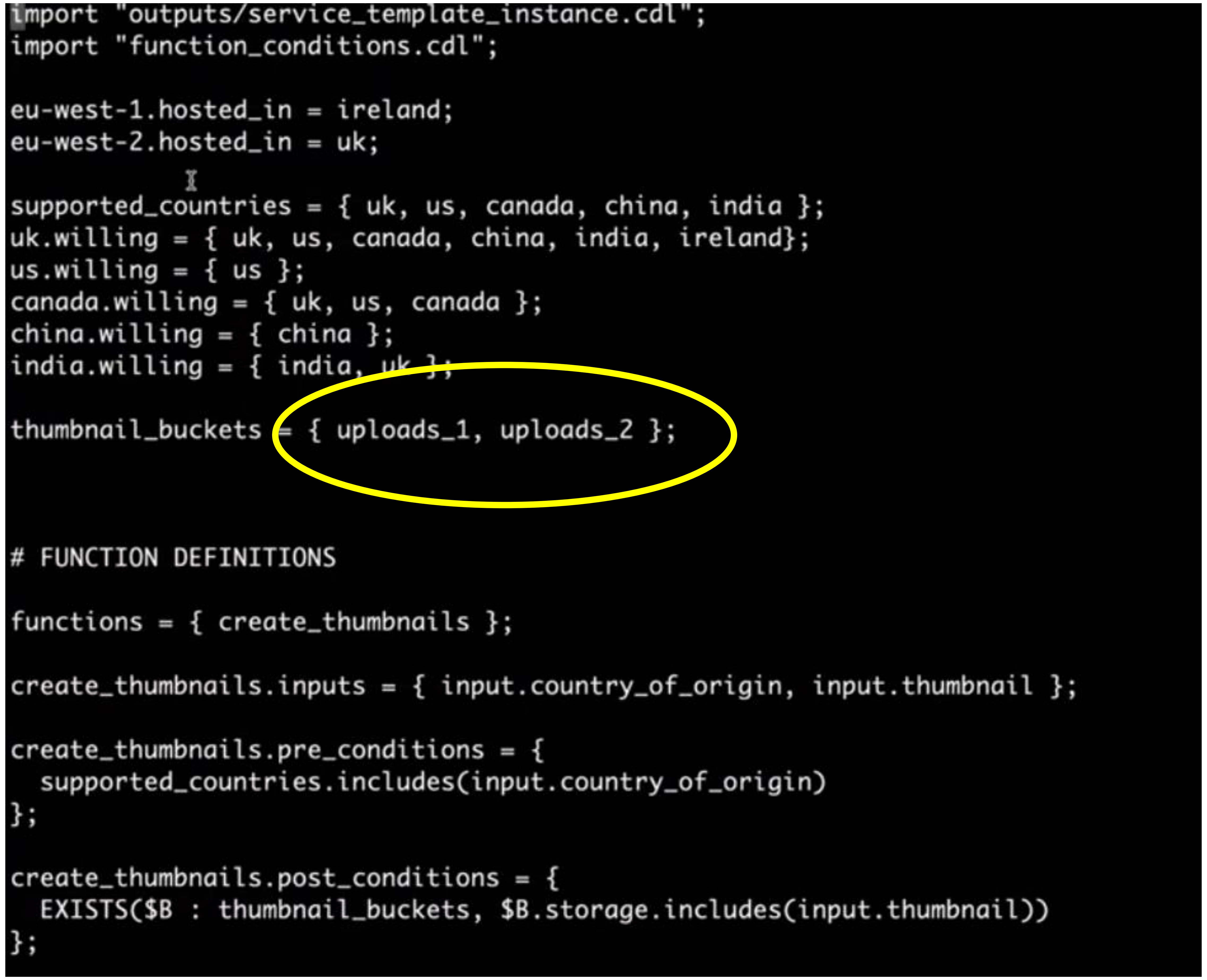
Constraints (expressed in CDL):

- When create_thumbnail is called, the thumbnail should be stored in a bucket located in a country that the country of origin is willing to share with.
- The UK is willing to share with anyone.
- India is only willing to share with India and the US.
- The US is only willing to share with the US.
- China is only willing to share with China.

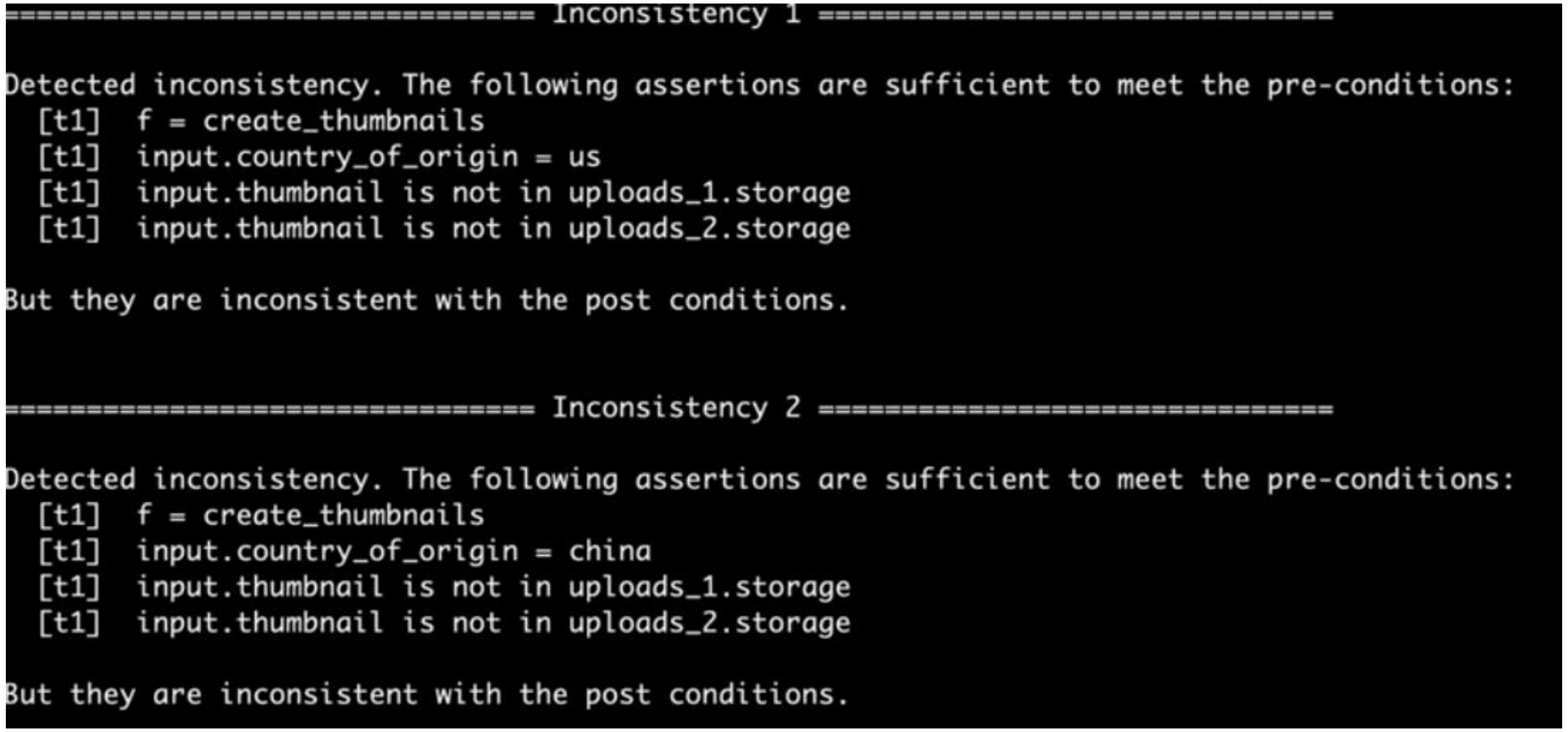
Inconsistencies:

1. country_of_origin = US
2. country_of_origin = China

Constraint Definition Language

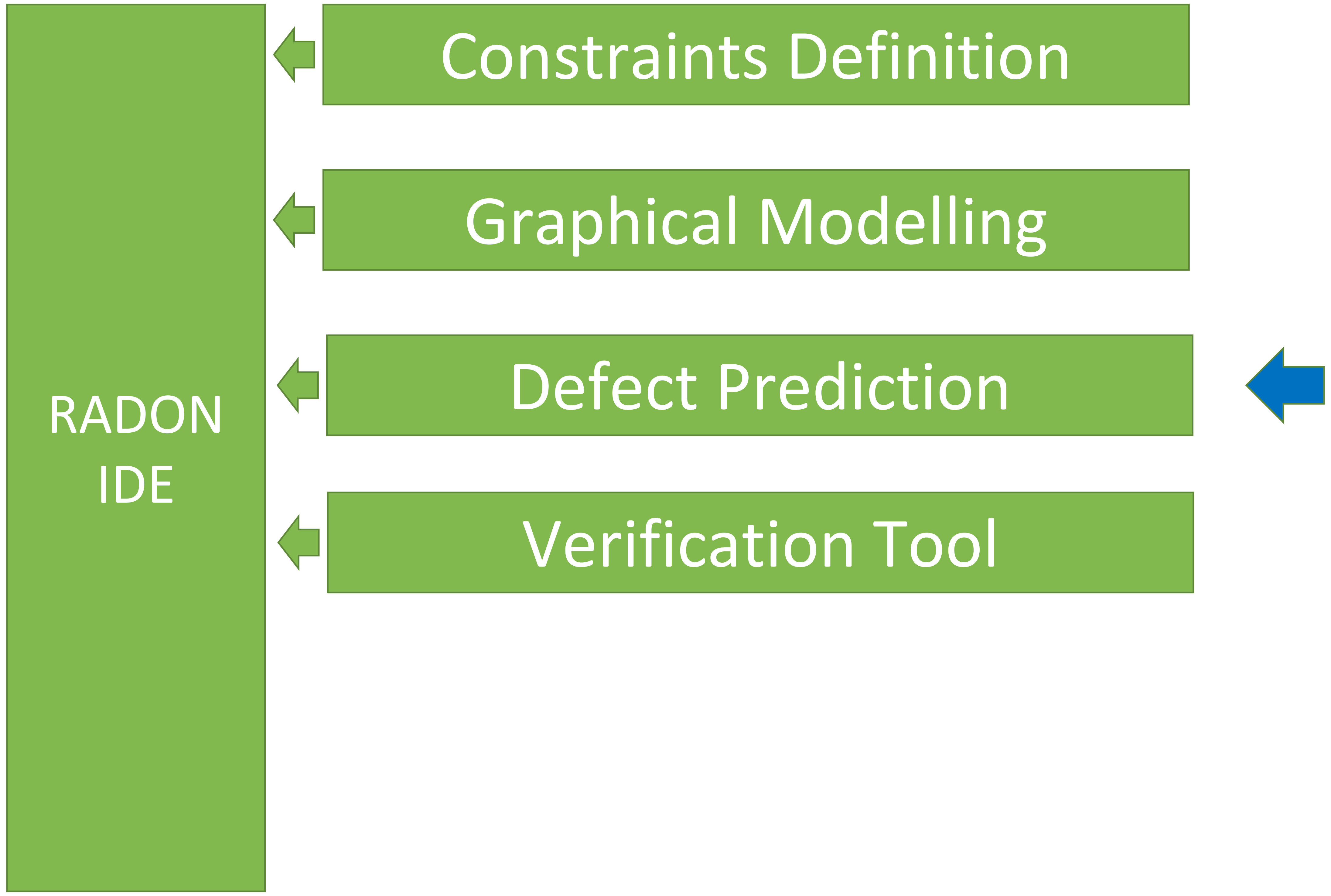


```
import "outputs/service_template_instance.cdl";
import "function_conditions.cdl";

eu-west-1.hosted_in = ireland;
eu-west-2.hosted_in = uk;

supported_countries = { uk, us, canada, china, india };
uk.willing = { uk, us, canada, china, india, ireland};
us.willing = { us };
canada.willing = { uk, us, canada };
china.willing = { china };
india.willing = { india, uk };

thumbnail_buckets = { uploads_1, uploads_2 };


# FUNCTION DEFINITIONS

functions = { create_thumbnails };

create_thumbnails.inputs = { input.country_of_origin, input.thumbnail };

create_thumbnails.pre_conditions = {
  supported_countries.includes(input.country_of_origin)
};

create_thumbnails.post_conditions = {
  EXISTS($B : thumbnail_buckets, $B.storage.includes(input.thumbnail))
};
```



```
================================ Inconsistency 1 ================================

Detected inconsistency. The following assertions are sufficient to meet the pre-conditions:
  [t1]  f = create_thumbnails
  [t1]  input.country_of_origin = us
  [t1]  input.thumbnail is not in uploads_1.storage
  [t1]  input.thumbnail is not in uploads_2.storage

But they are inconsistent with the post conditions.


================================ Inconsistency 2 ================================

Detected inconsistency. The following assertions are sufficient to meet the pre-conditions:
  [t1]  f = create_thumbnails
  [t1]  input.country_of_origin = china
  [t1]  input.thumbnail is not in uploads_1.storage
  [t1]  input.thumbnail is not in uploads_2.storage

But they are inconsistent with the post conditions.
```

RADON Framework Overview



RADON IDE

Constraints Definition

Graphical Modelling

Defect Prediction

Verification Tool

Design-to-runtime defects / anti-patterns analysis

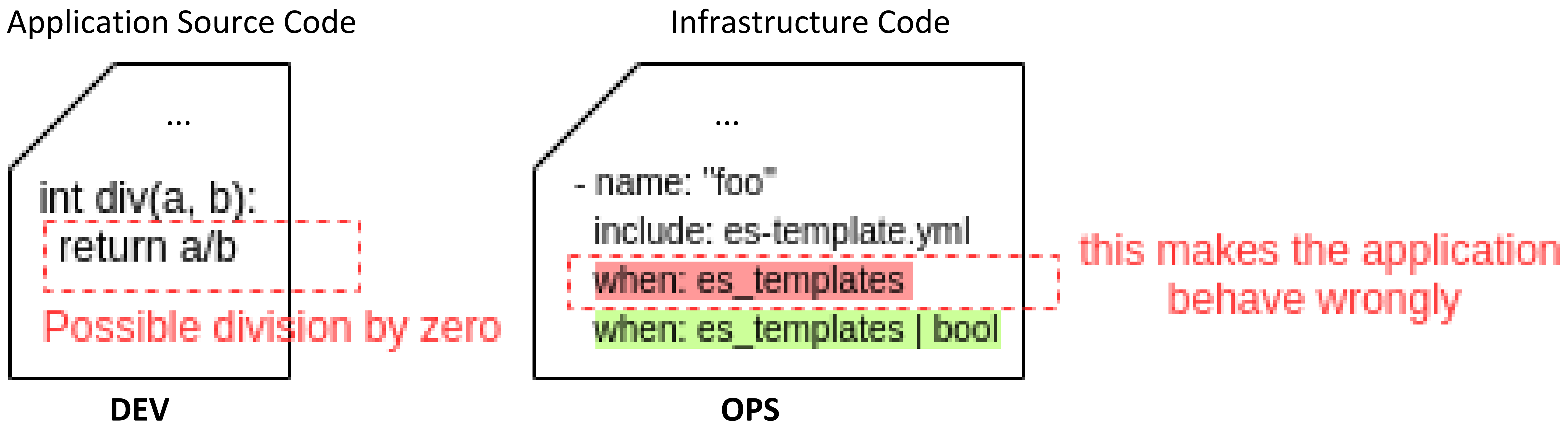
Defect Prediction Tool

Why?

"Infrastructure-as-code (IaC) ⇒ managing and provisioning compute datacenters through machine-readable definition files"

Cit. TOSCA Simple Profile Yaml v1.3, CSD2

- As any other **source code artifact**, **IaC** files may contain **defects** that can preclude their correct functioning and operations;



Application Source Code
Infrastructure Code
Possible division by zero
this makes the application behave wrongly
DEV
OPS

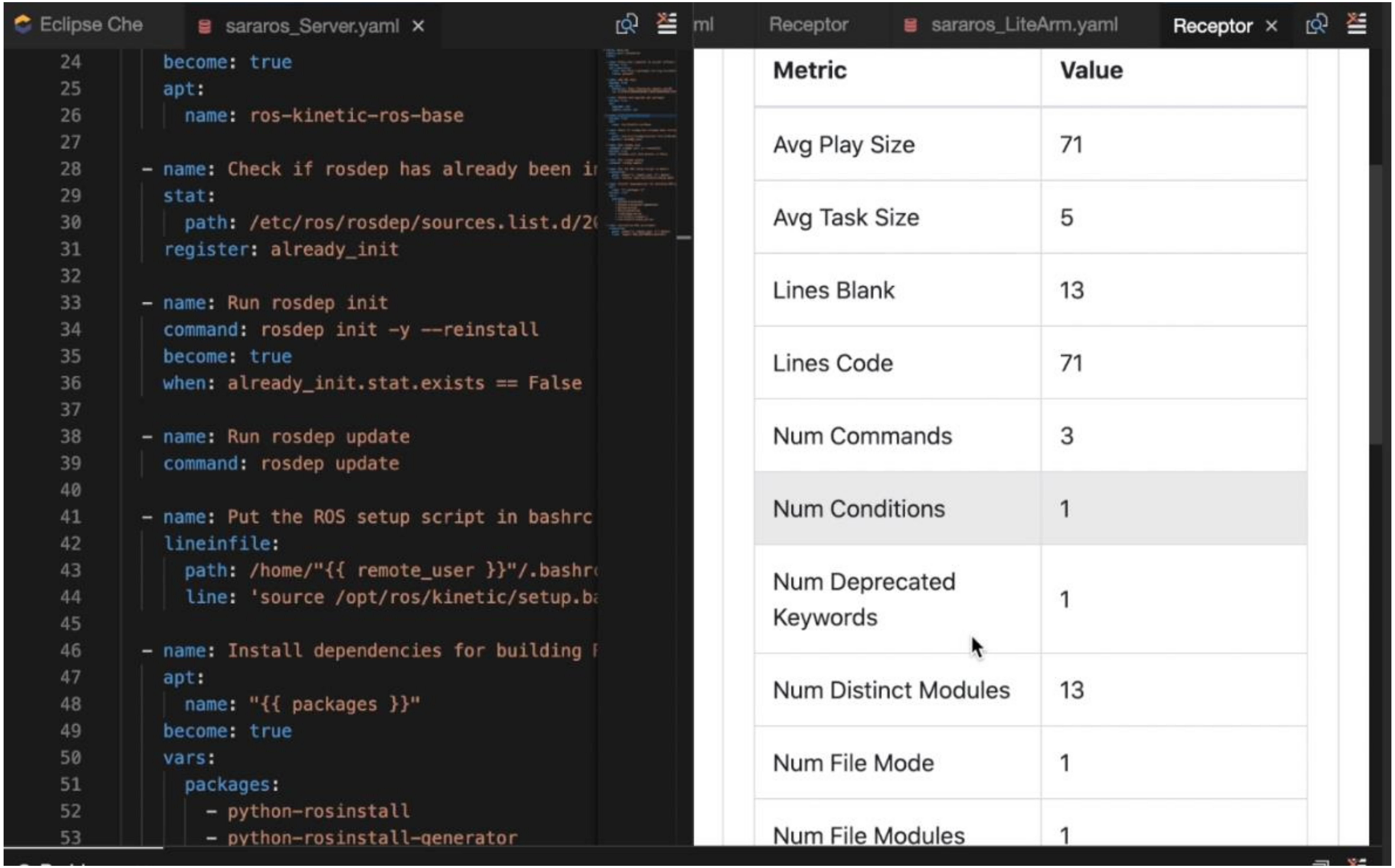
```
...
int div(a, b):
  return a/b
```

```
...
- name: "foo"
  include: es-template.yml
  when: es_templates
  when: es_templates | bool
```

- The tool is intended for **detecting defect-prone IaC blueprints** at the end of a release cycle;

- Defect-Prediction from Dev. source-code is well-established in the use of Machine-Learning techniques:
  - **Scripts** prone to contain **imperfections** or deficiencies cause them **not to meet their requirements or specifications**;
  - **Metrics** identify such **qualities**, so that **smells** or **bug-proneness** can be **detected** and possibly repaired;

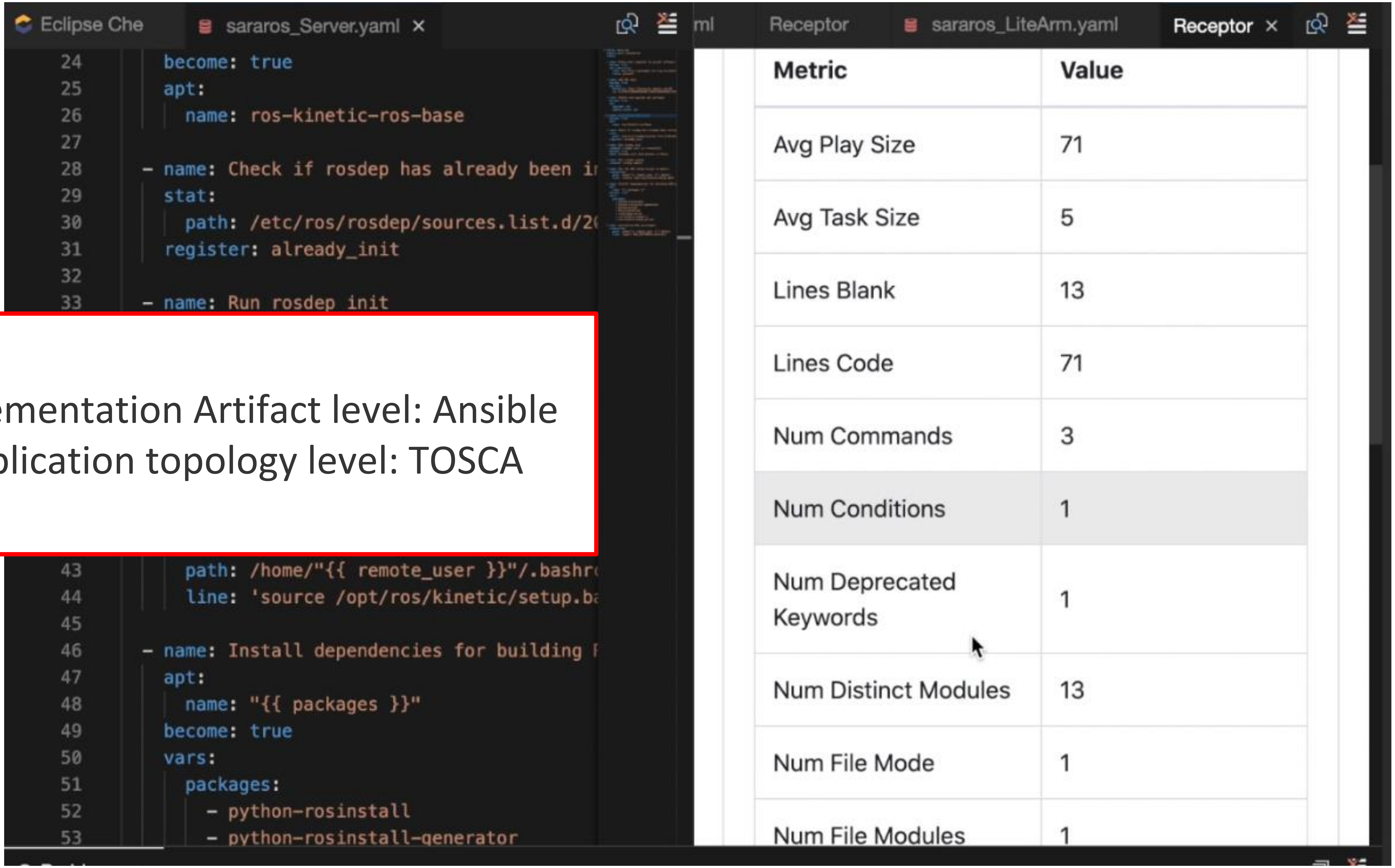
Defect Prediction Metrics



```
    become: true
    apt:
      name: ros-kinetic-ros-base

  - name: Check if rosdep has already been i
    stat:
      path: /etc/ros/rosdep/sources.list.d/2
    register: already_init

  - name: Run rosdep init
    command: rosdep init -y --reinstall
    become: true
    when: already_init.stat.exists == False

  - name: Run rosdep update
    command: rosdep update

  - name: Put the ROS setup script in bashrc
    lineinfile:
      path: /home/"{{ remote_user }}"/.bashr
      line: 'source /opt/ros/kinetic/setup.b

  - name: Install dependencies for building
    apt:
      name: "{{ packages }}"
    become: true
    vars:
      packages:
        - python-rosinstall
        - python-rosinstall-generator
```

Metric	Value
Avg Play Size	71
Avg Task Size	5
Lines Blank	13
Lines Code	71
Num Commands	3
Num Conditions	1
Num Deprecated Keywords	1
Num Distinct Modules	13
Num File Mode	1
Num File Modules	1

Defect Prediction Metrics



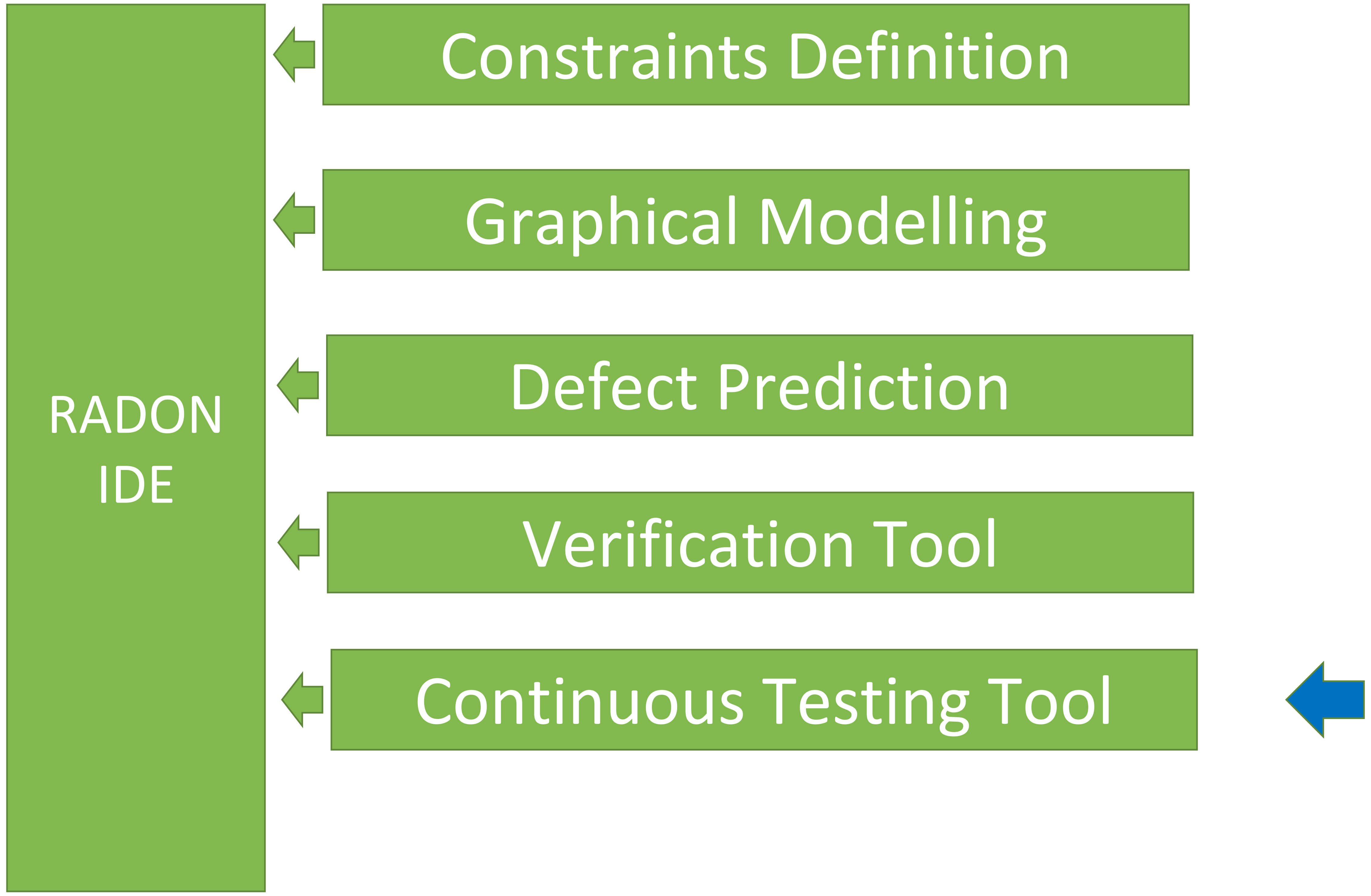
```
  become: true
  apt:
    name: ros-kinetic-ros-base

- name: Check if rosdep has already been i
  stat:
    path: /etc/ros/rosdep/sources.list.d/2
  register: already_init

- name: Run rosdep init
```

```
    path: /home/"{{ remote_user }}"/.bashr
    line: 'source /opt/ros/kinetic/setup.ba

- name: Install dependencies for building
  apt:
    name: "{{ packages }}"
  become: true
  vars:
    packages:
      - python-rosinstall
      - python-rosinstall-generator
```

Metric	Value
Avg Play Size	71
Avg Task Size	5
Lines Blank	13
Lines Code	71
Num Commands	3
Num Conditions	1
Num Deprecated Keywords	1
Num Distinct Modules	13
Num File Mode	1
Num File Modules	1

Implementation Artifact level: Ansible
Application topology level: TOSCA

RADON Framework Overview



RADON IDE

Constraints Definition

Graphical Modelling

Defect Prediction

Verification Tool

Continuous Testing Tool

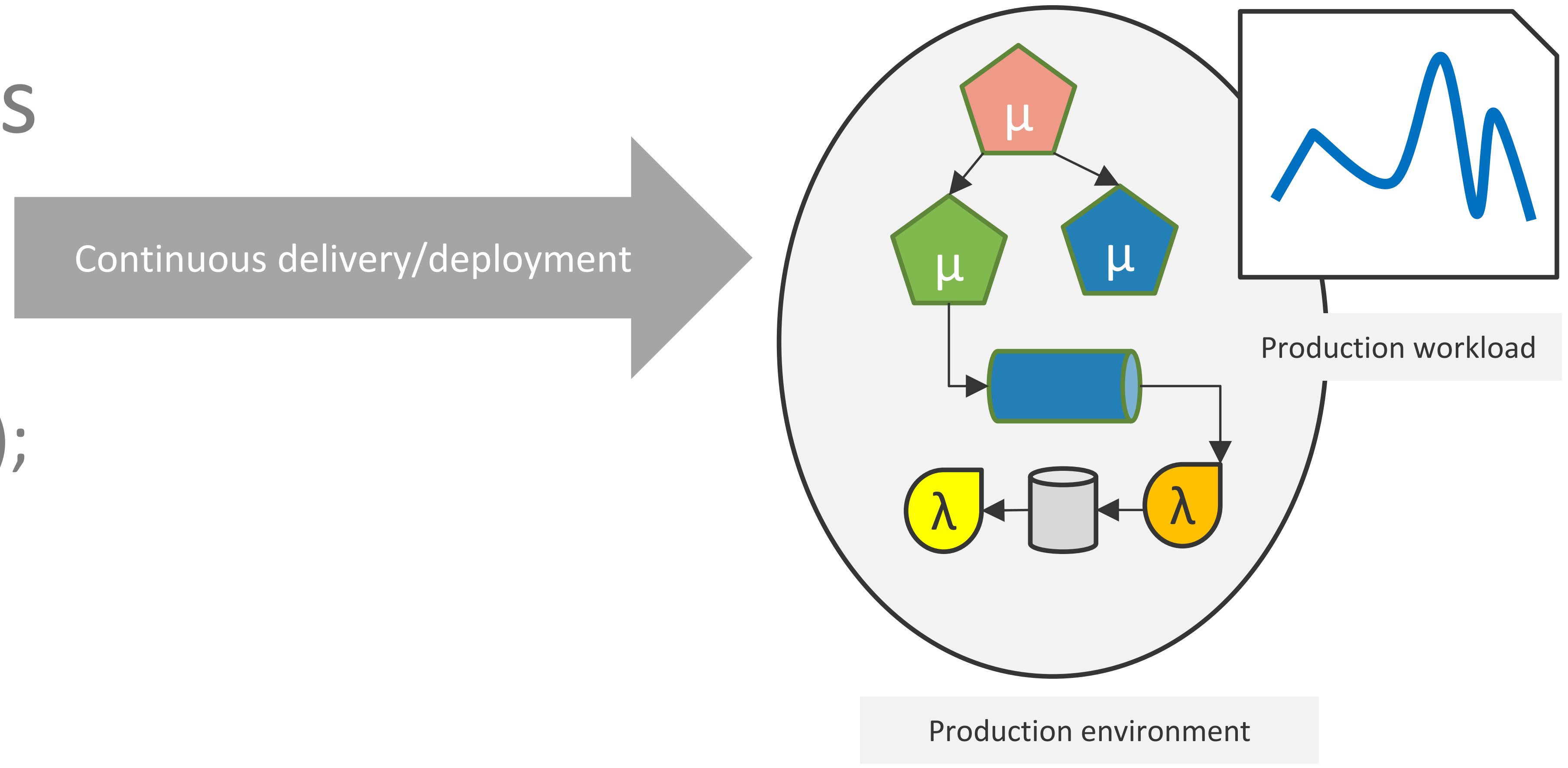
End-to-end pipelines testing;
DevOps closing-the-loop;

Continuous Testing

Why?

- Testing is key to assess **functional** and **non-functional** properties (e.g., performance):
  - different **scopes**: FaaS functions, microservices, and data pipelines;
  - different test **levels** (unit, integration, system testing);

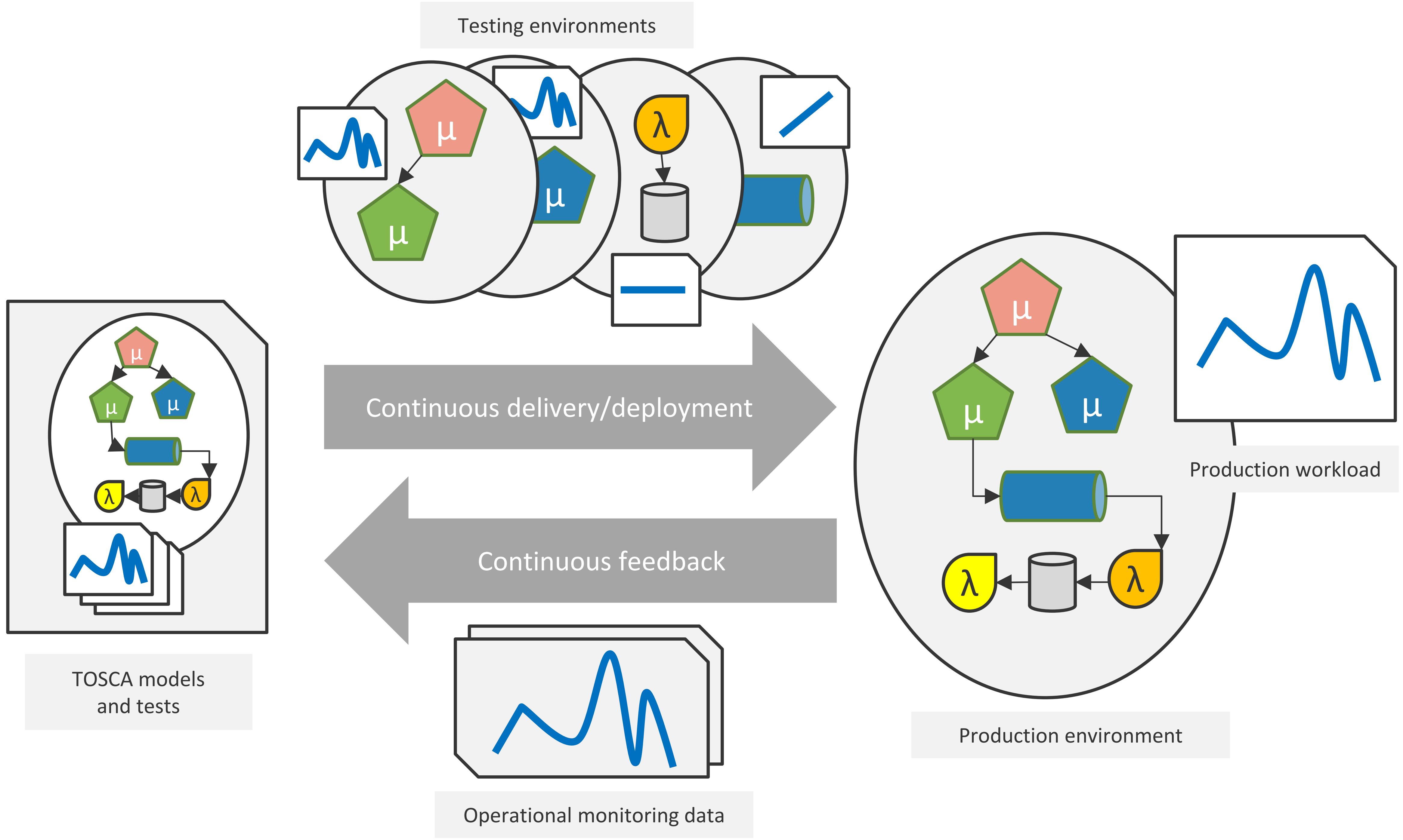
- Testing is **not** a one-time and manual activity but requires
  - **continuity** (on every CI/CD execution);
  - **automation** (e.g., test artifacts generation), and
  - **selection** (e.g., tailoring to workload scenarios and functions/microservices);



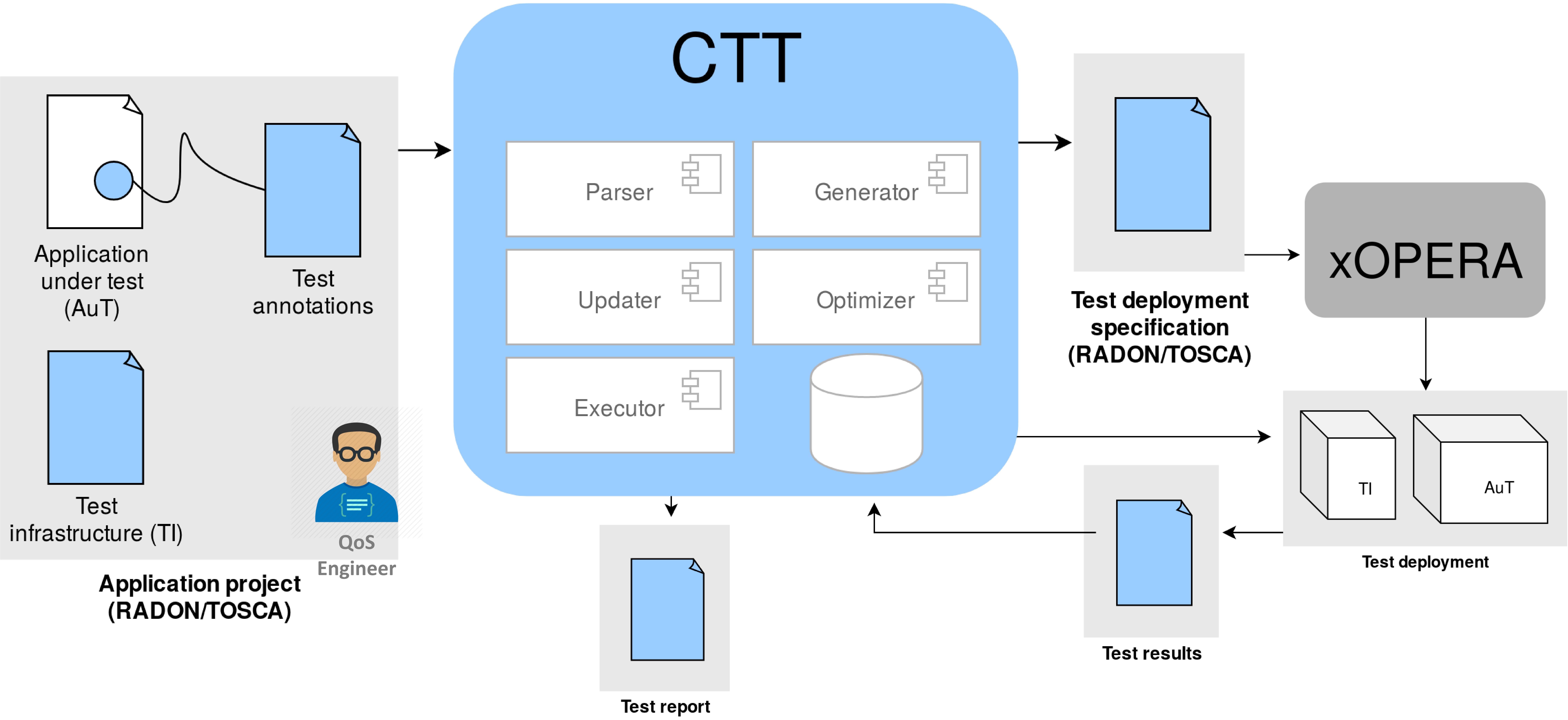
- Selected research challenges:
  - Frequent **changes** of the application and the operational profile;
  - **Conflict**: fast release cycles vs. time-consuming (e.g., performance and scalability) test runs;
    - Scalability of data pipelines (CTT data pipeline module) and microservices;
  - Testing in cloud infrastructures (e.g., **repeatability**, **access to metrics**);

Continuous Testing

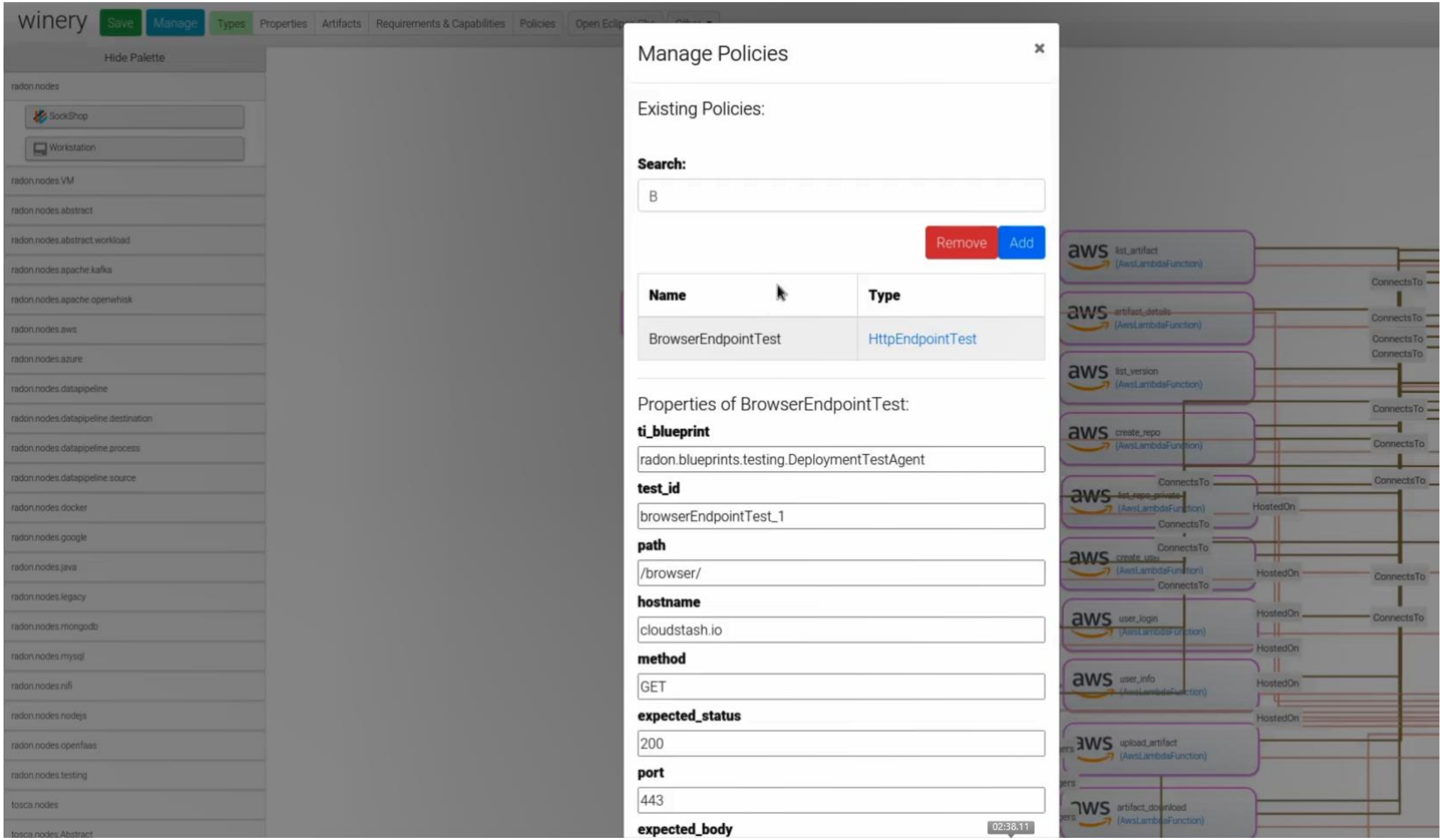
- Functionalities grouped into 3 usage scenarios:
  - Test case definition;
  - Test execution;
  - Test maintenance;
- CTT modules
  - Microservices/FaaS
  - Data pipelines
- Usage:
  - Standalone tool (open-source);
  - Invocation via the RADON IDE or CI/CD;



Continuous Testing



Continuous Testing Tool & GMT


winery
Save
Manage
Types
Properties
Artifacts
Requirements & Capabilities
Policies
Open Eclip
Hide Palette
radon.nodes
SockShop
Workstation
radon.nodes.VM
radon.nodes.abstract
radon.nodes.abstract.workload
radon.nodes.apache.kafka
radon.nodes.apache.openwhisk
radon.nodes.aws
radon.nodes.azure
radon.nodes.datapipeline
radon.nodes.datapipeline.destination
radon.nodes.datapipeline.process
radon.nodes.datapipeline.source
radon.nodes.docker
radon.nodes.google
radon.nodes.java
radon.nodes.legacy
radon.nodes.mongodb
radon.nodes.mysql
radon.nodes.nifi
radon.nodes.nodejs
radon.nodes.openfaas
radon.nodes.testing
tosca.nodes
tosca.nodes.Abstract
aws list_artifact (AwsLambdaFunction)
aws artifact_details (AwsLambdaFunction)
aws list_version (AwsLambdaFunction)
aws create_repo (AwsLambdaFunction)
aws list_repo_private (AwsLambdaFunction)
aws create_user (AwsLambdaFunction)
aws user_login (AwsLambdaFunction)
aws user_info (AwsLambdaFunction)
aws upload_artifact (AwsLambdaFunction)
aws artifact_download (AwsLambdaFunction)
ConnectsTo
HostedOn
02:38.11
×
Remove
Add

Manage Policies

Existing Policies:

Search:

B

Name	Type
BrowserEndpointTest	HttpEndpointTest

Properties of BrowserEndpointTest:

ti_blueprint

radon.blueprints.testing.DeploymentTestAgent

test_id

browserEndpointTest_1

path

/browser/

hostname

cloudstash.io

method

GET

expected_status

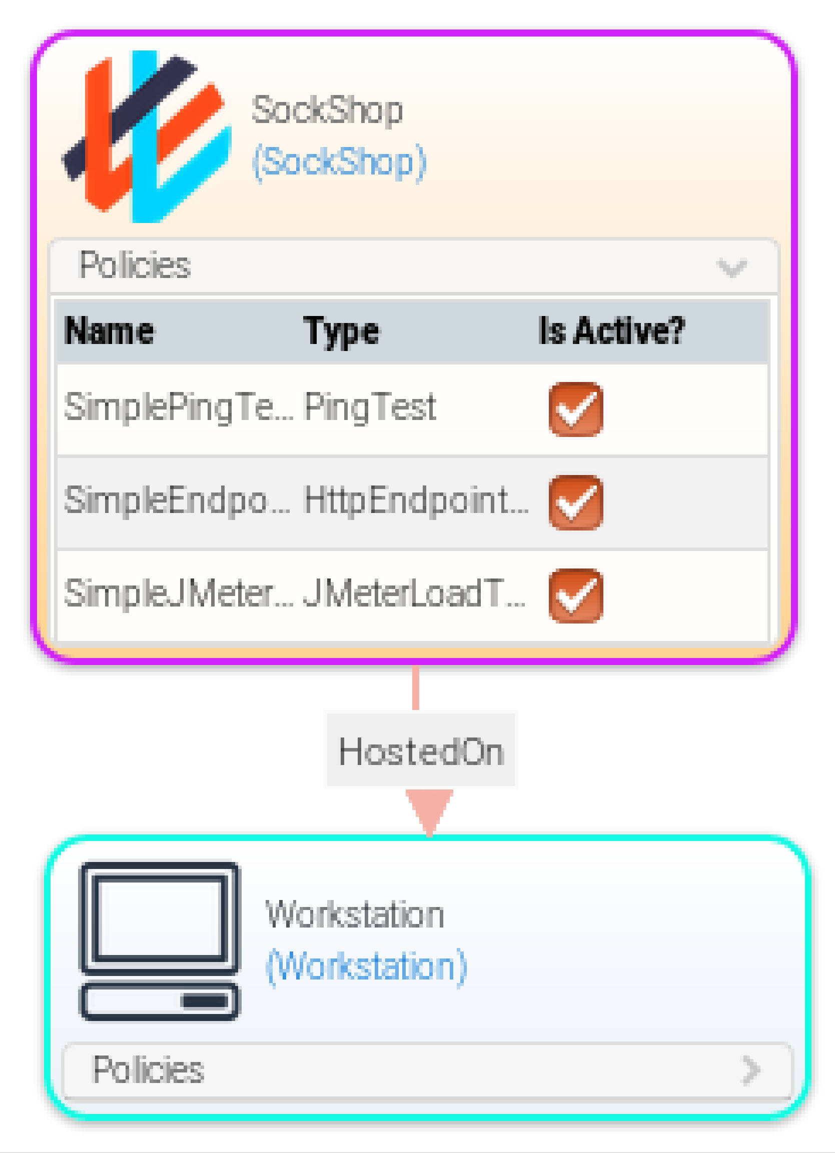
200

port

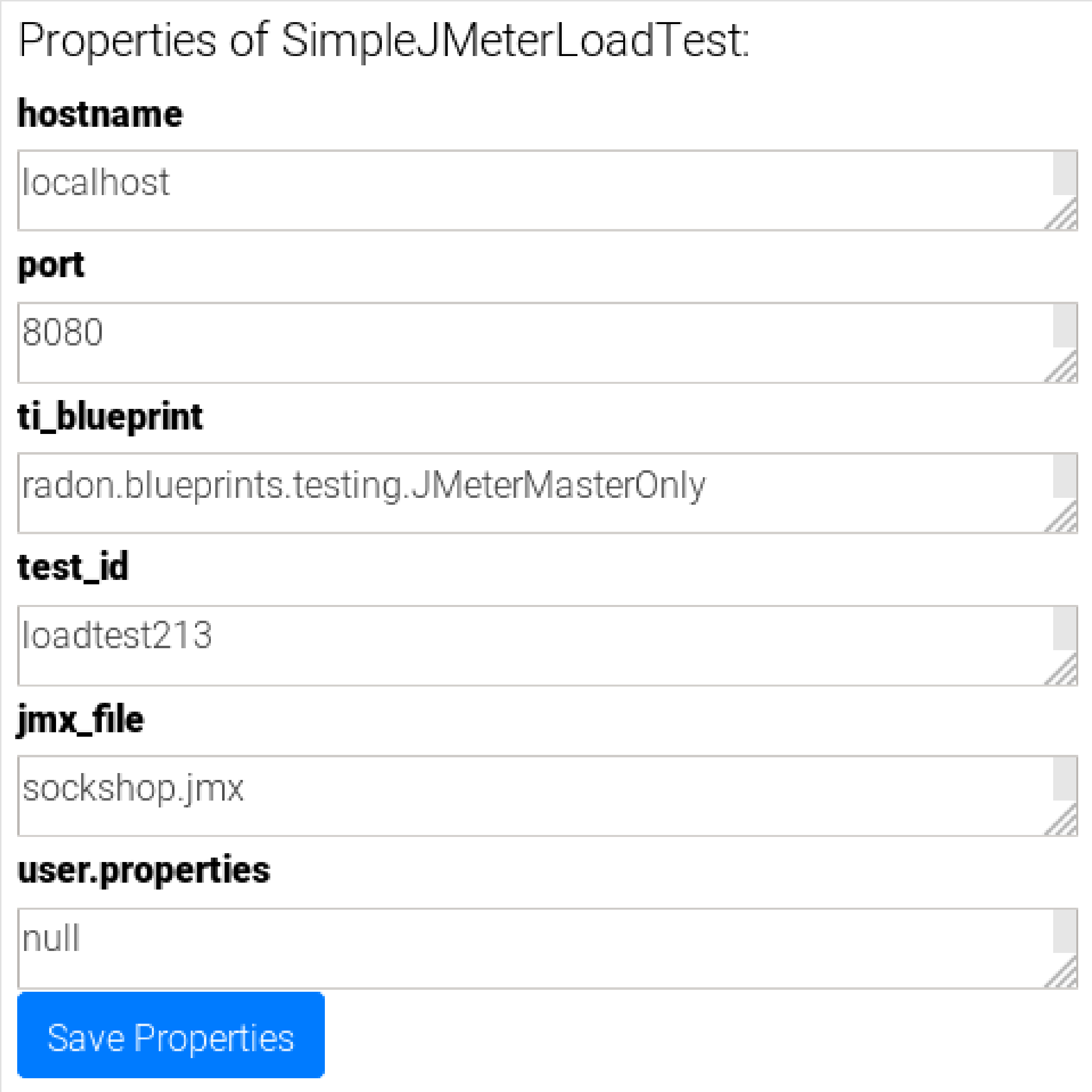
443

expected_body

CTT Modeling in GMT – SUT and TI



Name	Type	Is Active?
SimplePingTe...	PingTest	☑
SimpleEndpo...	HttpEndpoint...	☑
SimpleJMeter...	JMeterLoadT...	☑



Properties of SimpleJMeterLoadTest:

hostname

localhost

port

8080

ti_blueprint

radon.blueprints.testing.JMeterMasterOnly

test_id

loadtest213

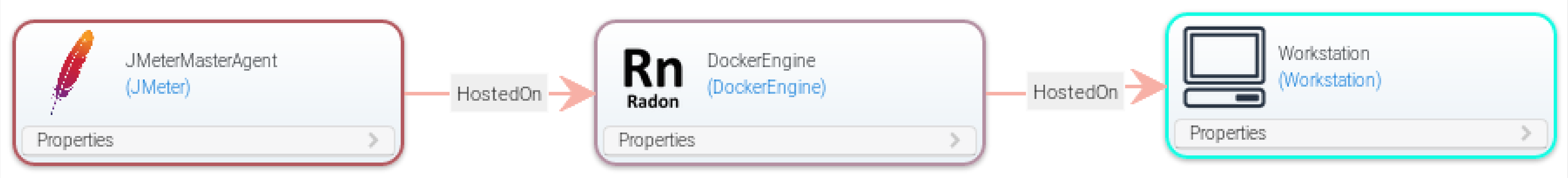
jmx_file

sockshop.jmx

user.properties

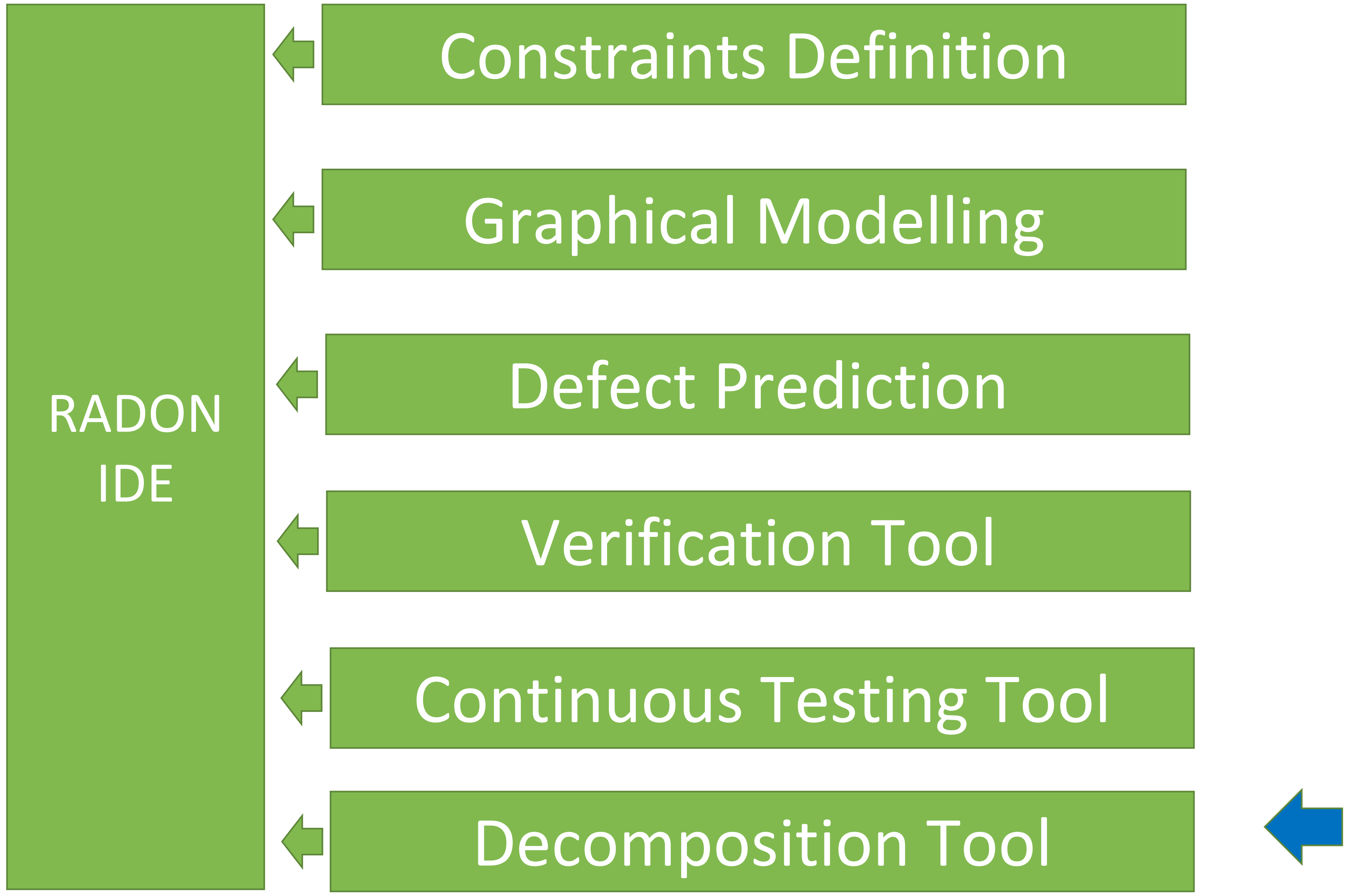
null

SUT



TI

RADON Framework Overview



Functional decomposition, model optimization

Decomposition Tool

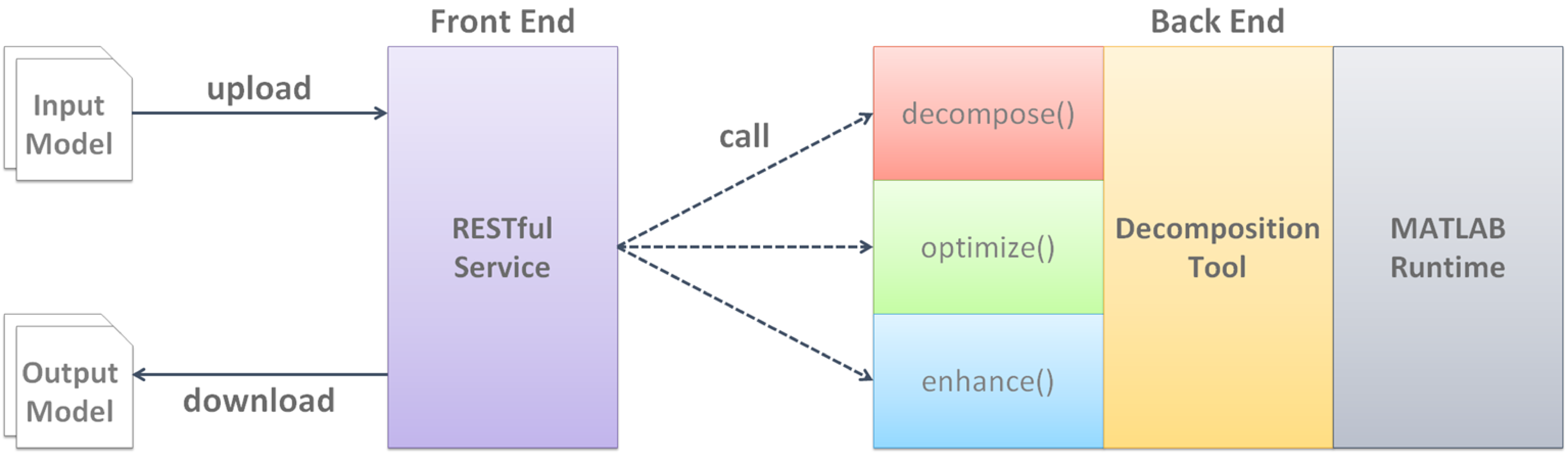
Why?

- Refactoring the architecture is never an easy job:
  - **granularity level**: coarse-grained, fine-grained and mixed-grain
  - **heterogeneity**: monoliths, microservices, serverless functions, object stores and data pipelines
  - **other considerations**: security and privacy

- It is also difficult to decide the **optimal** deployment scheme for the decomposed application (e.g. memory and concurrency of a serverless function):
  - **minimize** the **operating costs** on a target platform
  - **satisfy** the specified **performance requirements**

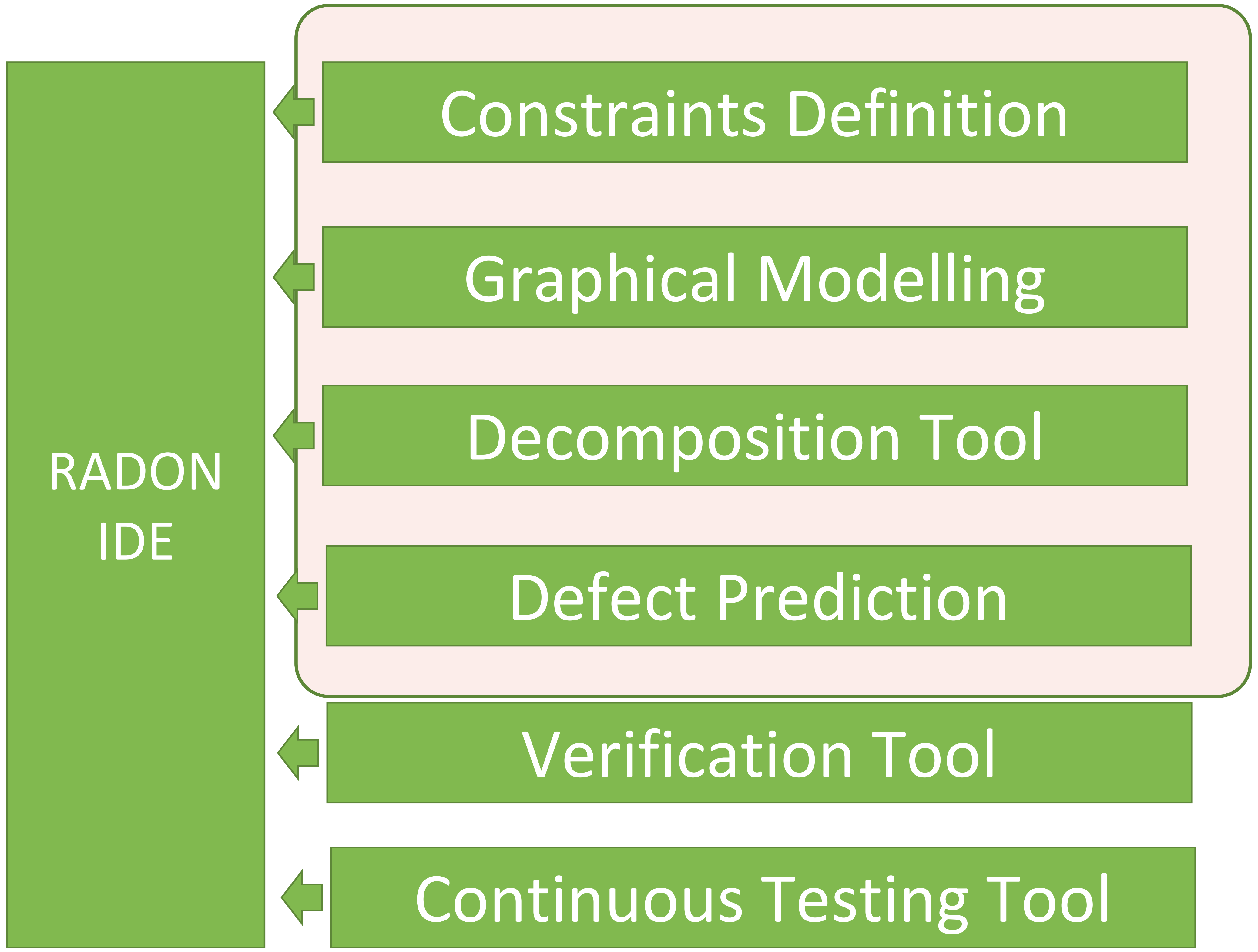
Decomposition Tool

- Three typical usage scenarios:
  - architecture decomposition;
  - deployment optimization;
  - accuracy enhancement (enable an **iterative DevOps** design lifecycle)



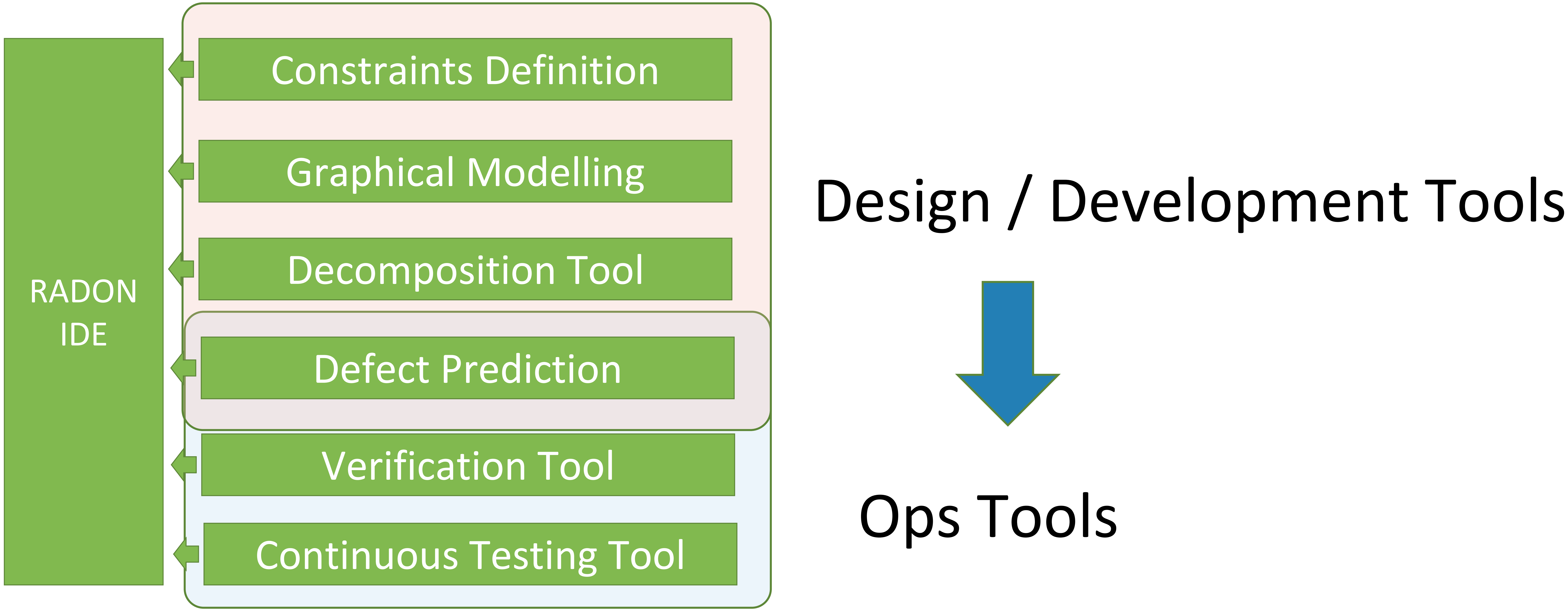
(See https://github.com/radon-h2020/decomposition-tool)

RADON Framework Overview

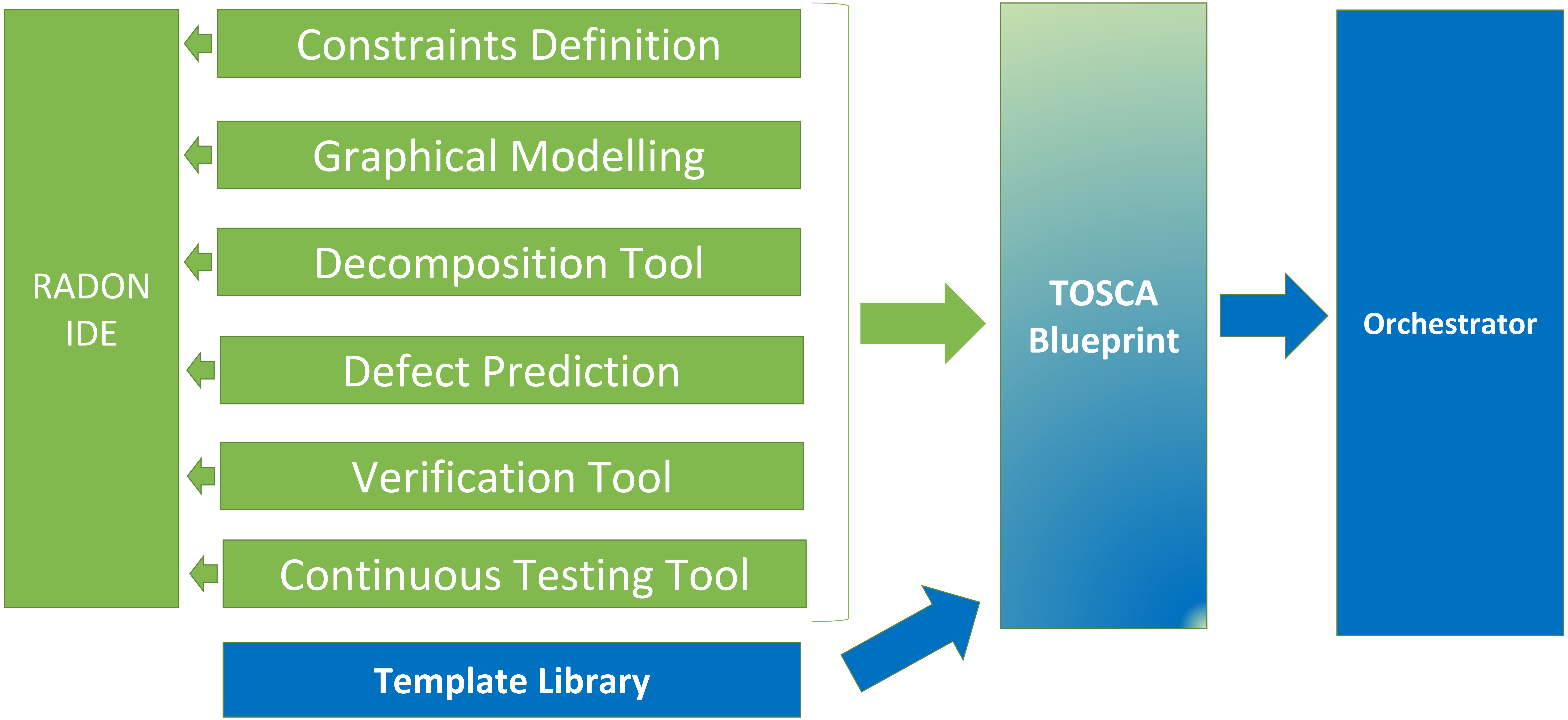


Design / Development Tools

RADON Framework Overview

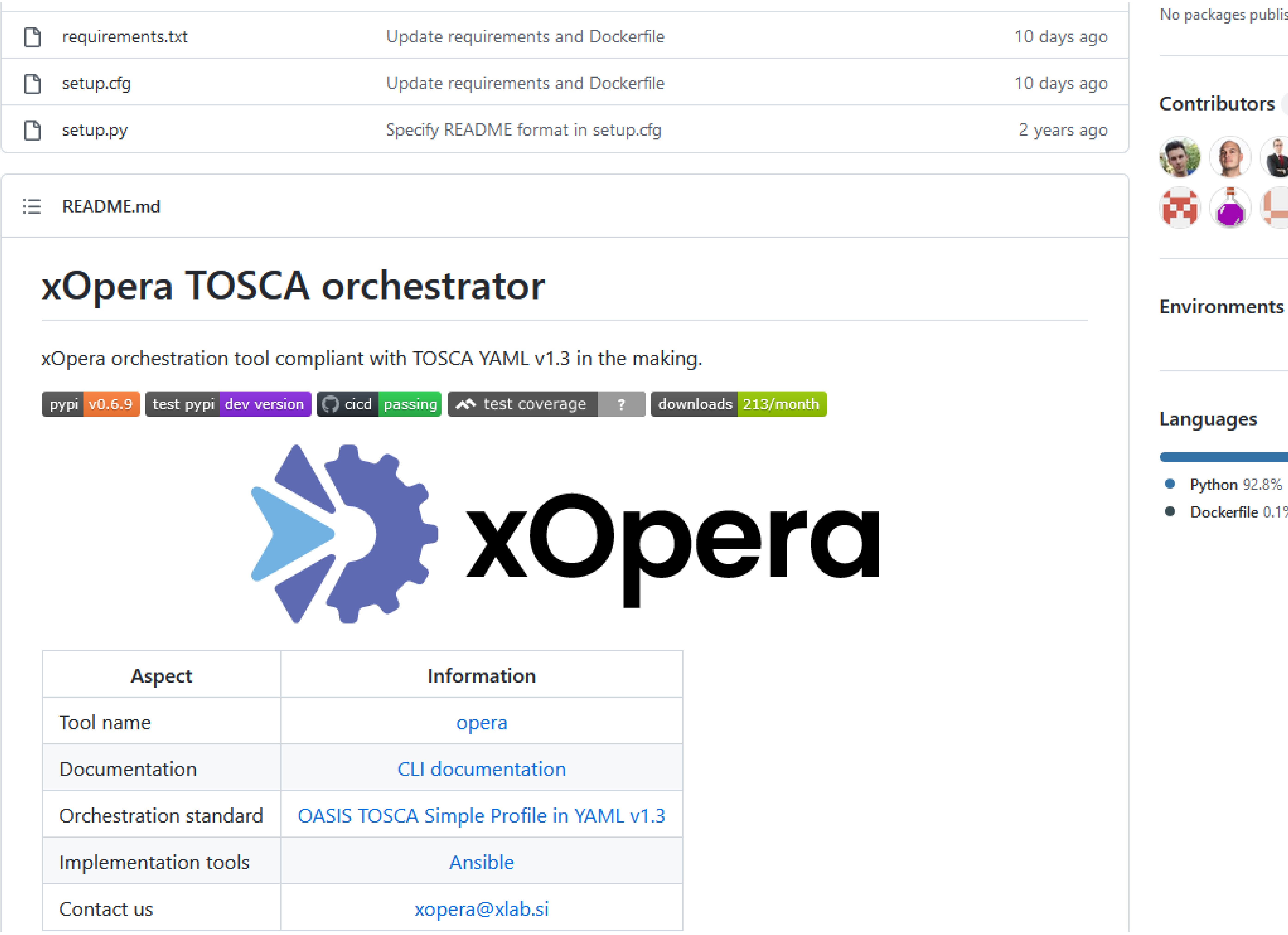


RADON Framework Overview



RADON Orchestrator

- Lightweight TOSCA orchestrator
- Ansible is used as orchestration actuators within the TOSCA interface operations
- Available as self-hosted CLI tool and SaaS offering



requirements.txt	Update requirements and Dockerfile	10 days ago
setup.cfg	Update requirements and Dockerfile	10 days ago
setup.py	Specify README format in setup.cfg	2 years ago

README.md

xOpera TOSCA orchestrator

xOpera orchestration tool compliant with TOSCA YAML v1.3 in the making.

Aspect	Information
Tool name	opera
Documentation	CLI documentation
Orchestration standard	OASIS TOSCA Simple Profile in YAML v1.3
Implementation tools	Ansible
Contact us	xopera@xlab.si

No packages publis

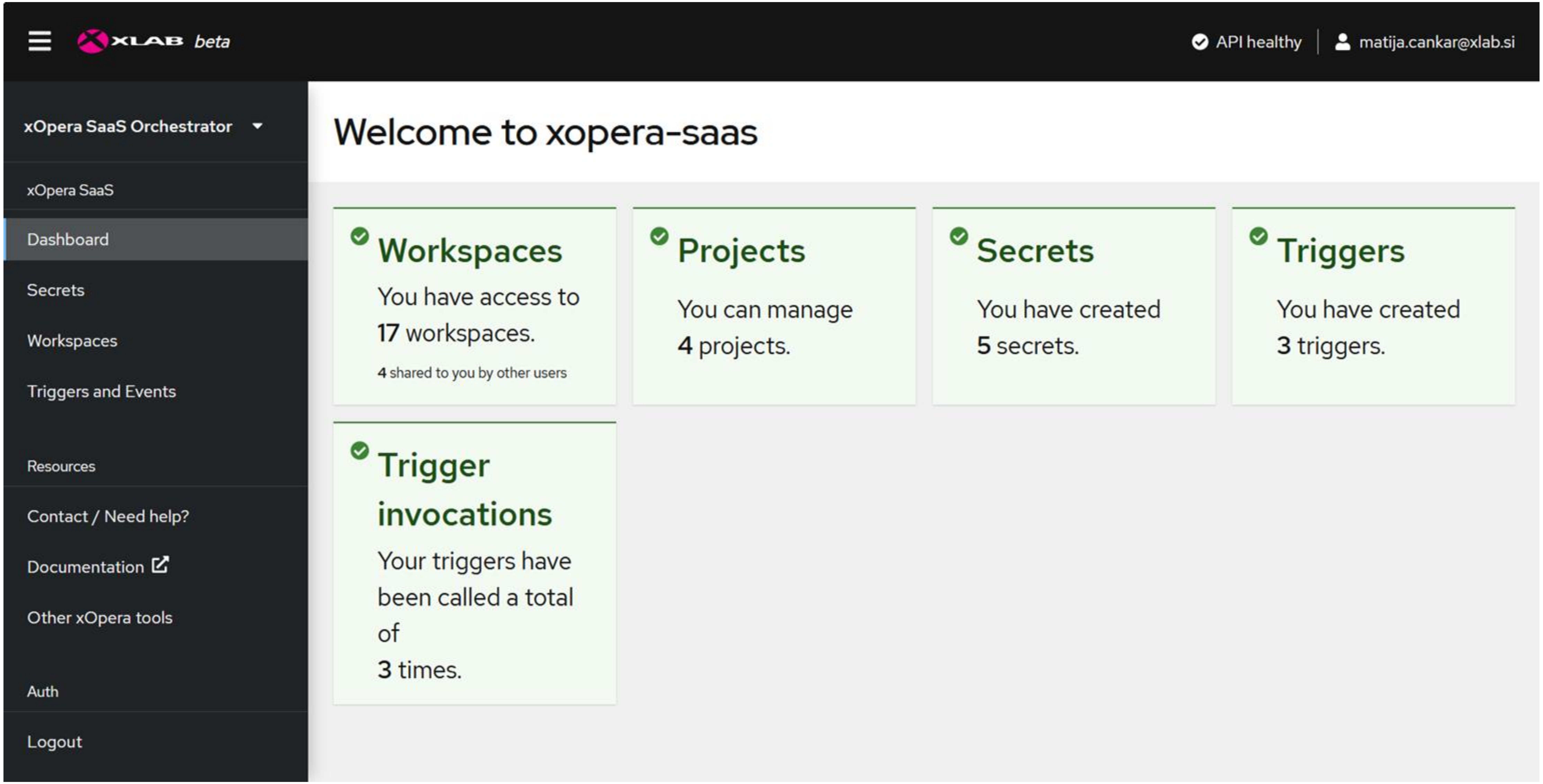
Contributors

Environments

Languages

- Python 92.8%
- Dockerfile 0.1

RADON Orchestrator



Welcome to xopera-saas

✓ Workspaces

You have access to 17 workspaces.

4 shared to you by other users

✓ Projects

You can manage 4 projects.

✓ Secrets

You have created 5 secrets.

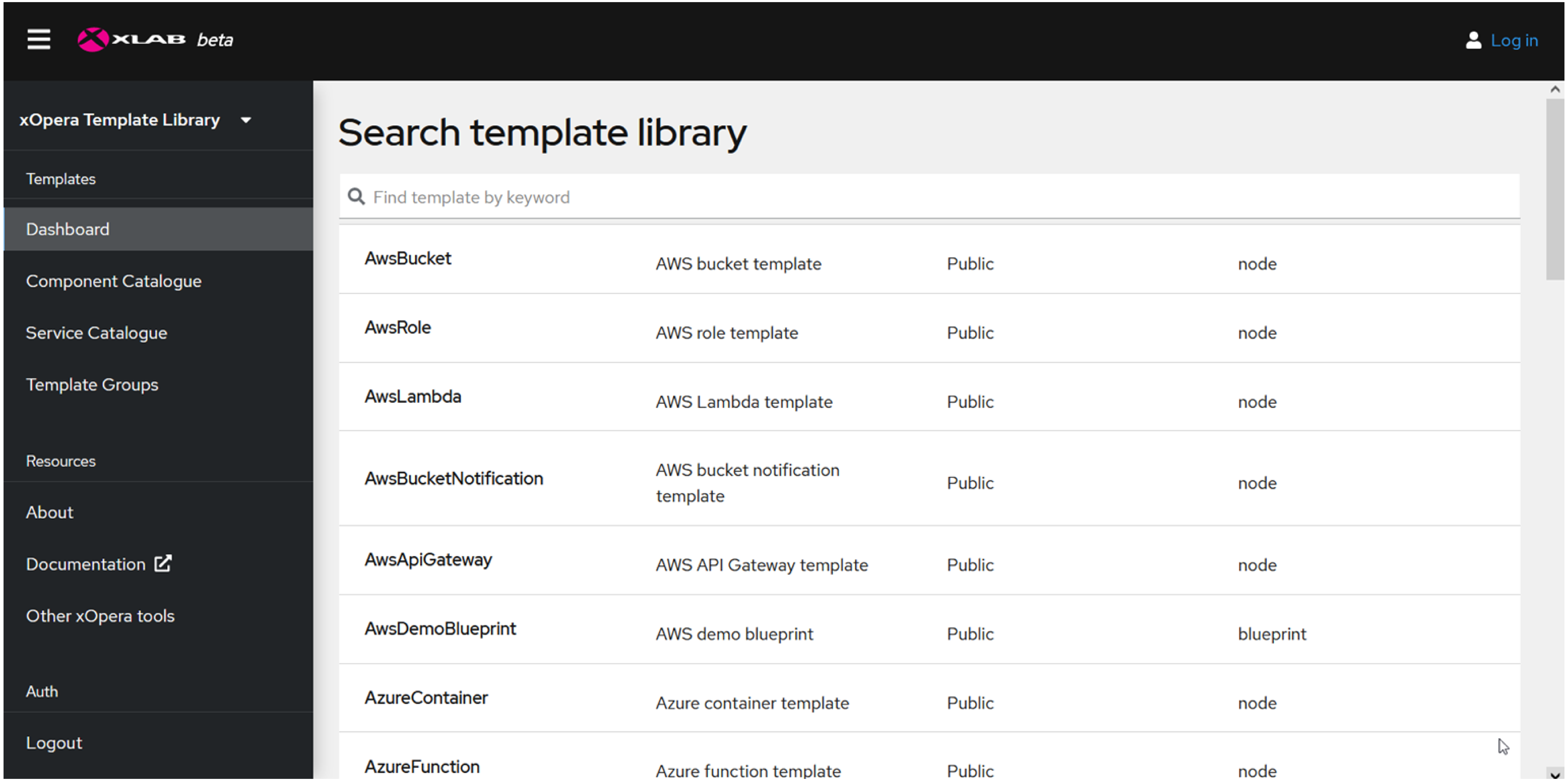
✓ Triggers

You have created 3 triggers.

✓ Trigger invocations

Your triggers have been called a total of 3 times.

RADON Template Library: Managed Models Repository

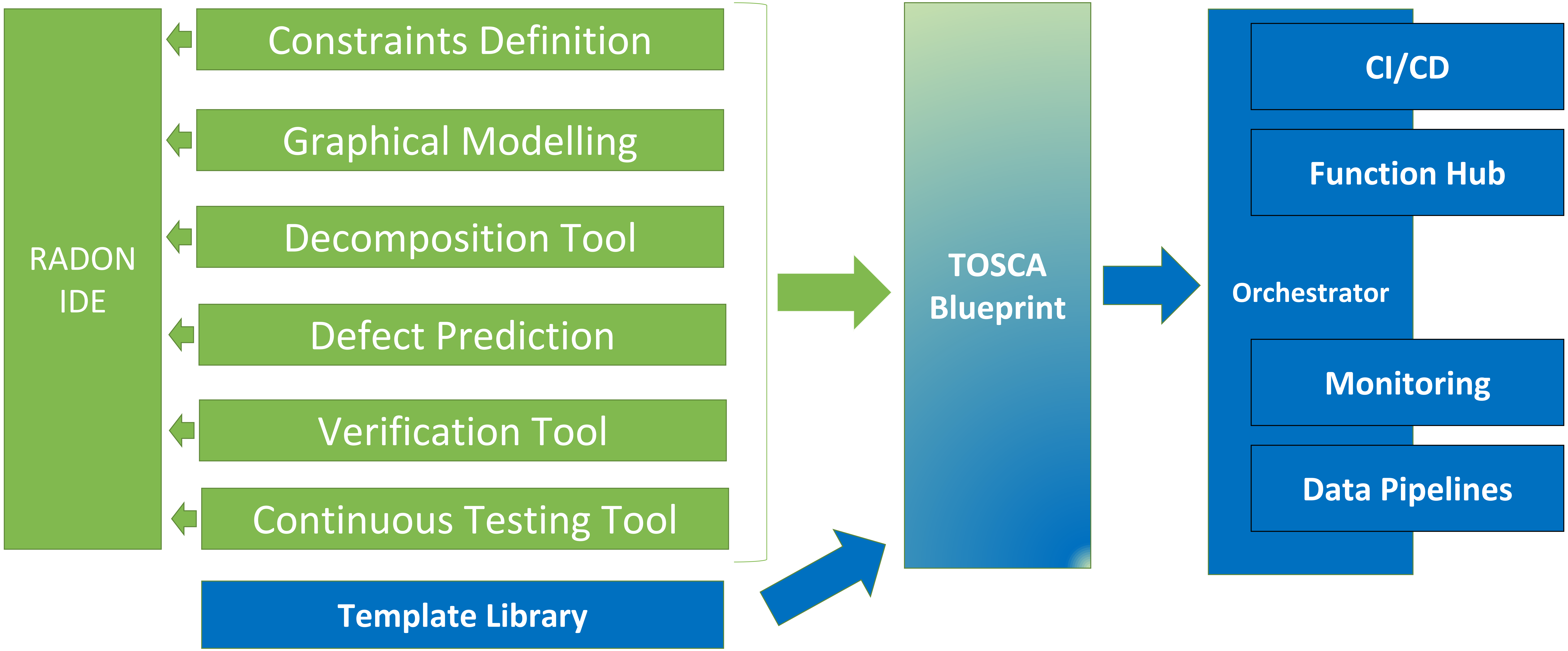


Search template library

Find template by keyword

AwsBucket	AWS bucket template	Public	node
AwsRole	AWS role template	Public	node
AwsLambda	AWS Lambda template	Public	node
AwsBucketNotification	AWS bucket notification template	Public	node
AwsApiGateway	AWS API Gateway template	Public	node
AwsDemoBlueprint	AWS demo blueprint	Public	blueprint
AzureContainer	Azure container template	Public	node
AzureFunction	Azure function template	Public	node

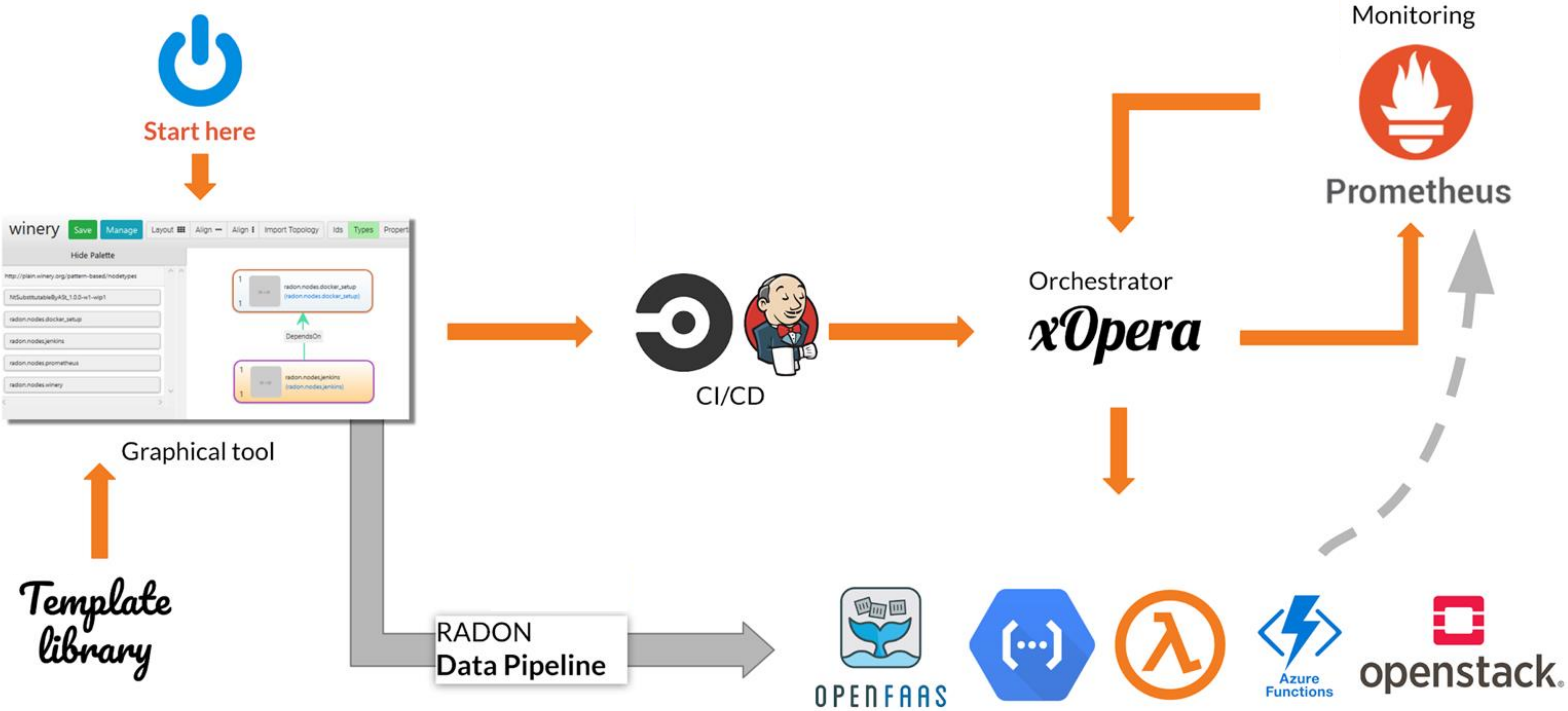
RADON Framework Overview



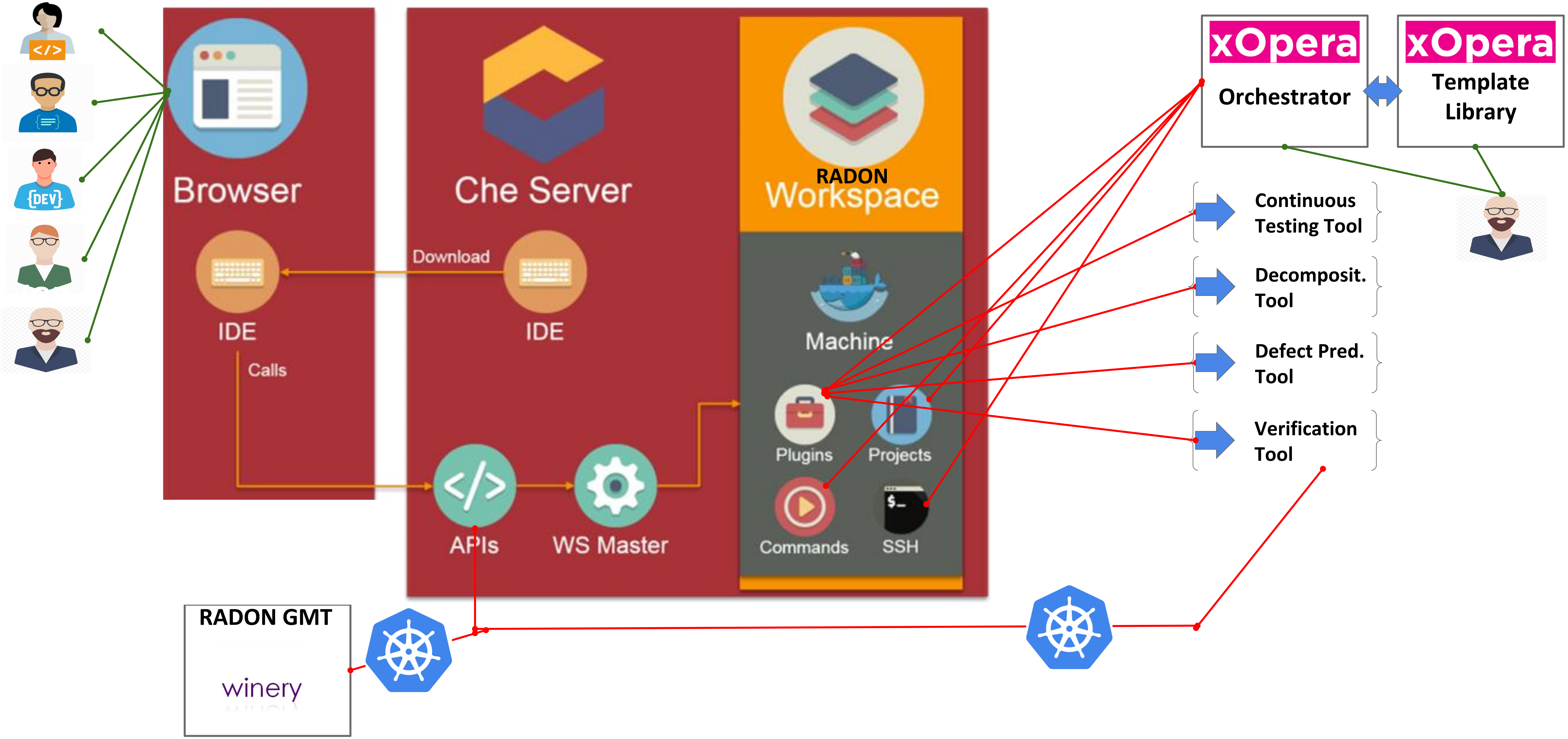
🟩 Design-time components

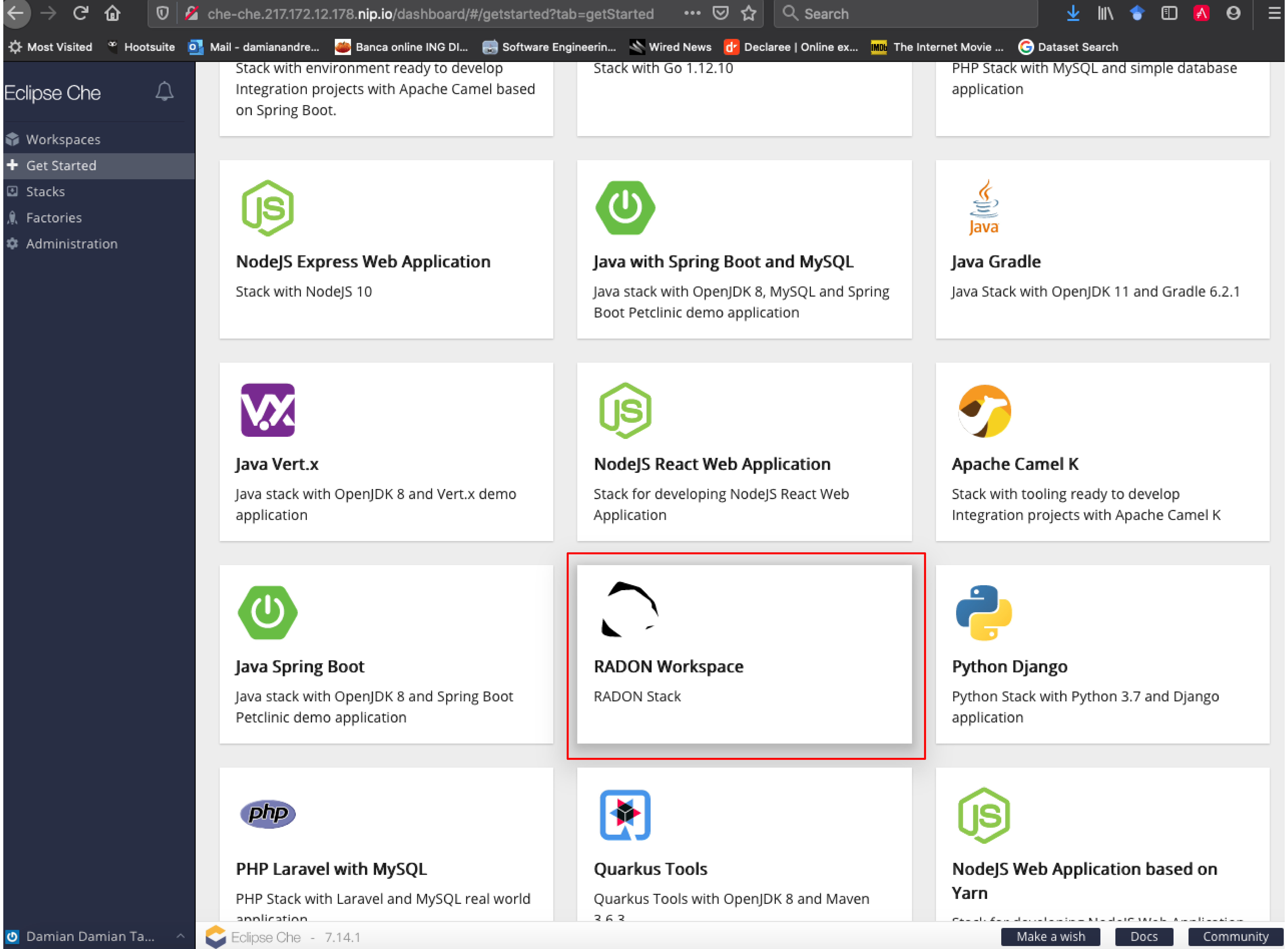
🟦 Runtime components

RADON Runtime Environment



RADON IDE: Overview





Stack with environment ready to develop Integration projects with Apache Camel based on Spring Boot.

Stack with Go 1.12.10

PHP Stack with MySQL and simple database application

NodeJS Express Web Application

Stack with NodeJS 10

Java with Spring Boot and MySQL

Java stack with OpenJDK 8, MySQL and Spring Boot Petclinic demo application

Java Gradle

Java Stack with OpenJDK 11 and Gradle 6.2.1

Java Vert.x

Java stack with OpenJDK 8 and Vert.x demo application

NodeJS React Web Application

Stack for developing NodeJS React Web Application

Apache Camel K

Stack with tooling ready to develop Integration projects with Apache Camel K

Java Spring Boot

Java stack with OpenJDK 8 and Spring Boot Petclinic demo application

RADON Workspace

RADON Stack

Python Django

Python Stack with Python 3.7 and Django application

PHP Laravel with MySQL

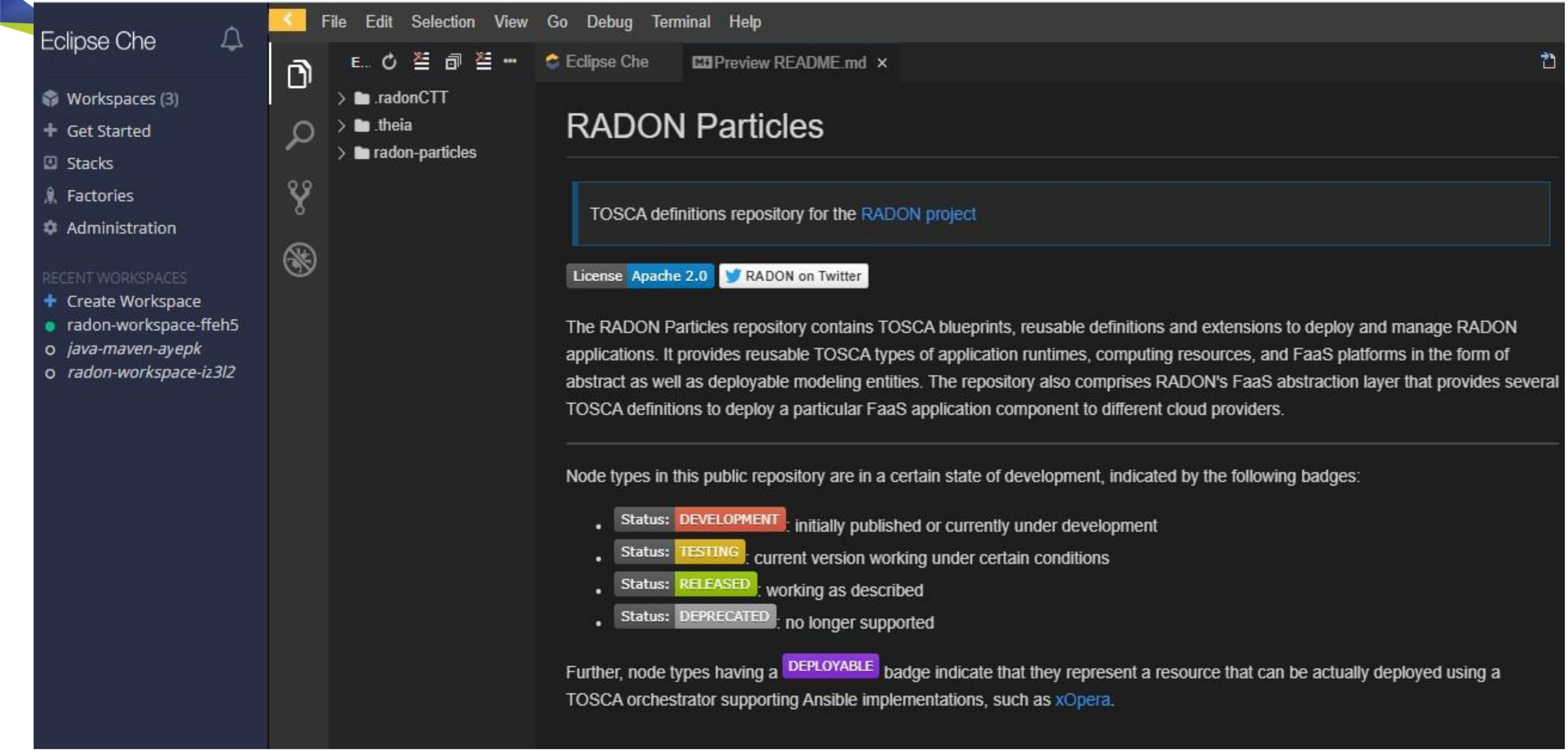
PHP Stack with Laravel and MySQL real world application

Quarkus Tools

Quarkus Tools with OpenJDK 8 and Maven 3.6.3

NodeJS Web Application based on Yarn

RADON IDE



RADON Particles

TOSCA definitions repository for the RADON project

License Apache 2.0 | RADON on Twitter

The RADON Particles repository contains TOSCA blueprints, reusable definitions and extensions to deploy and manage RADON applications. It provides reusable TOSCA types of application runtimes, computing resources, and FaaS platforms in the form of abstract as well as deployable modeling entities. The repository also comprises RADON's FaaS abstraction layer that provides several TOSCA definitions to deploy a particular FaaS application component to different cloud providers.

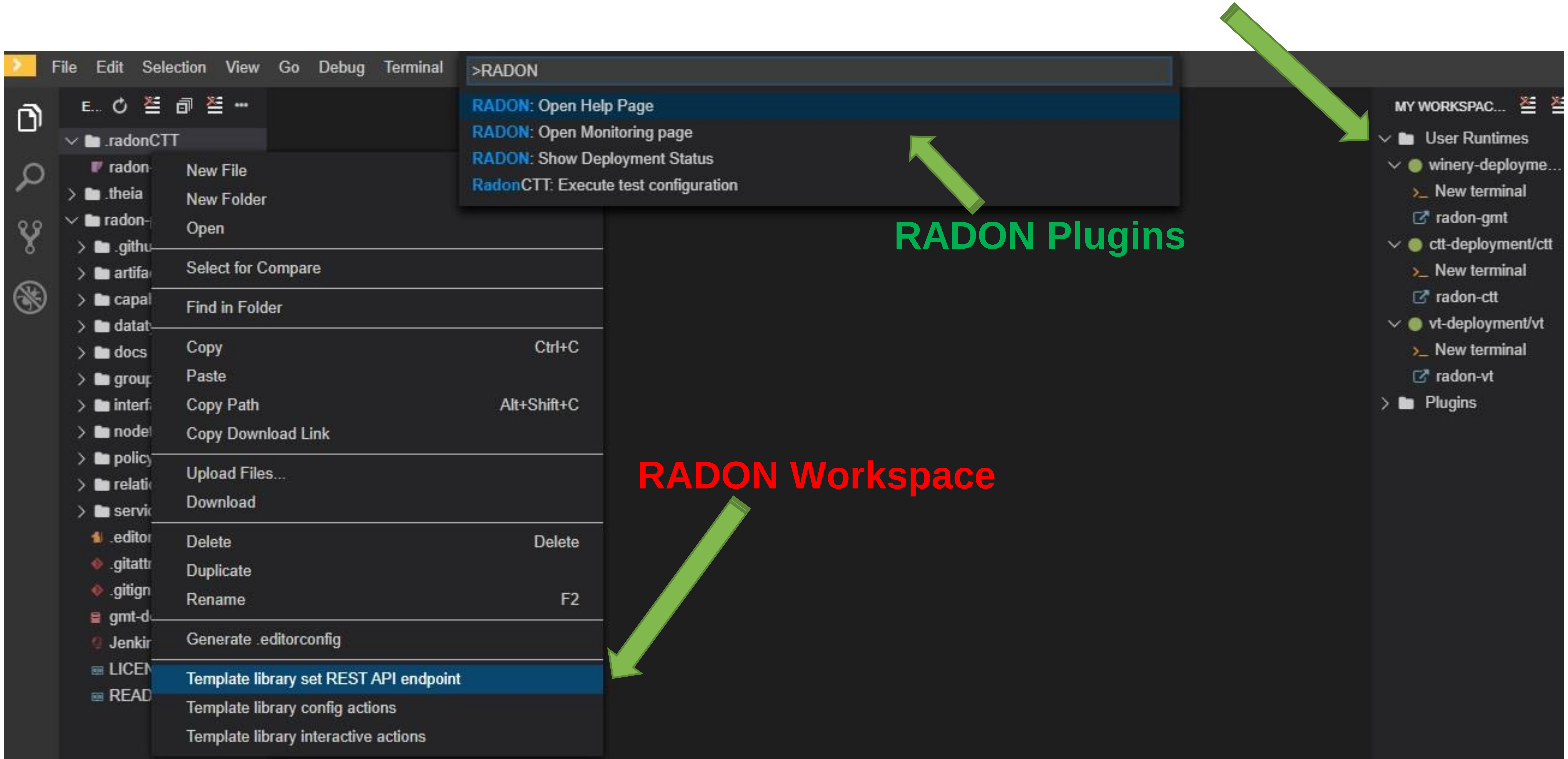
Node types in this public repository are in a certain state of development, indicated by the following badges:

- Status: DEVELOPMENT: initially published or currently under development
- Status: TESTING: current version working under certain conditions
- Status: RELEASED: working as described
- Status: DEPRECATED: no longer supported

Further, node types having a DEPLOYABLE badge indicate that they represent a resource that can be actually deployed using a TOSCA orchestrator supporting Ansible implementations, such as xOpera.

RADON IDE Overview

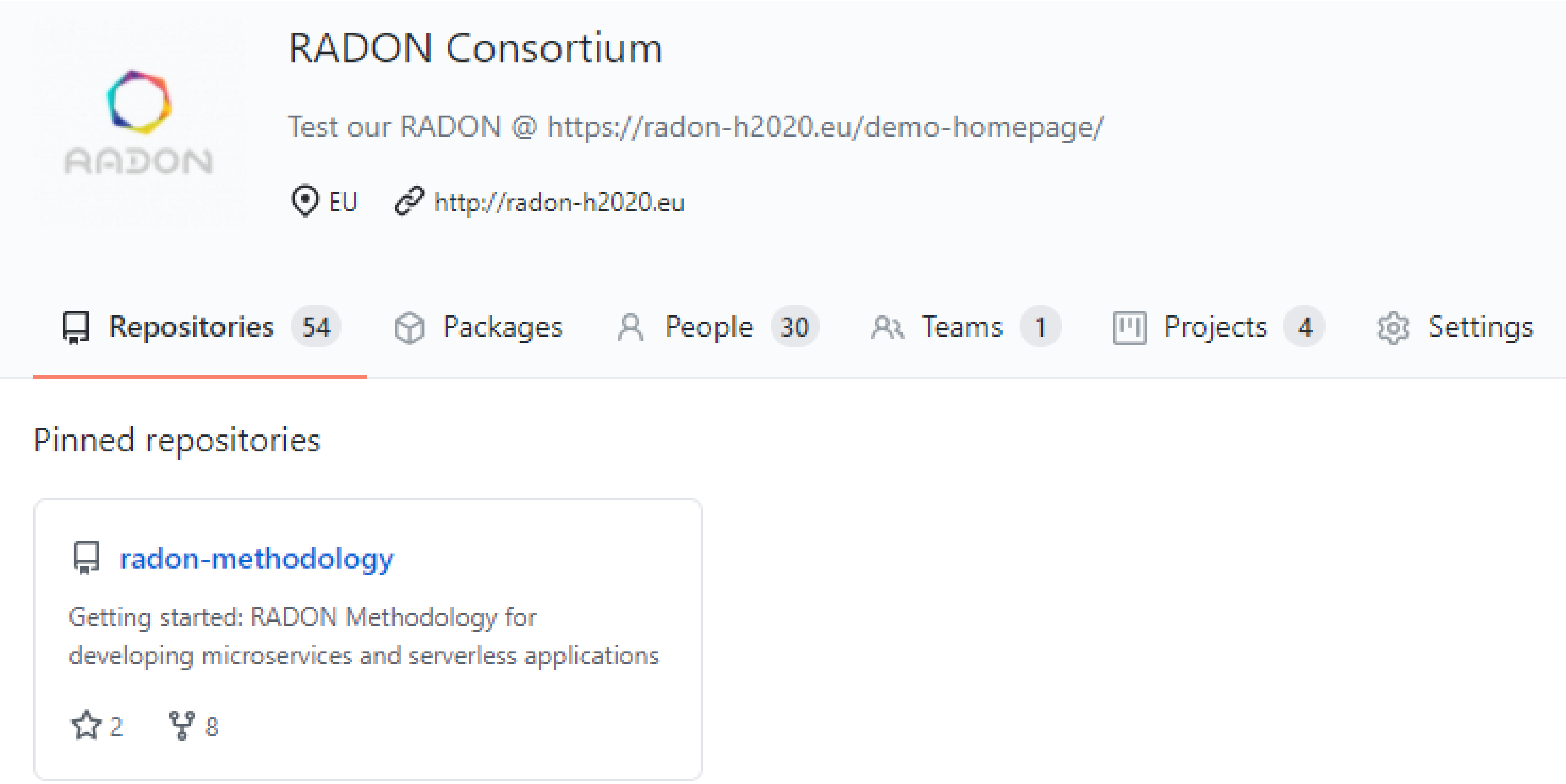
RADON Kubernetes components



RADON Plugins

RADON Workspace

More About RADON



RADON Consortium

Test our RADON @ https://radon-h2020.eu/demo-homepage/

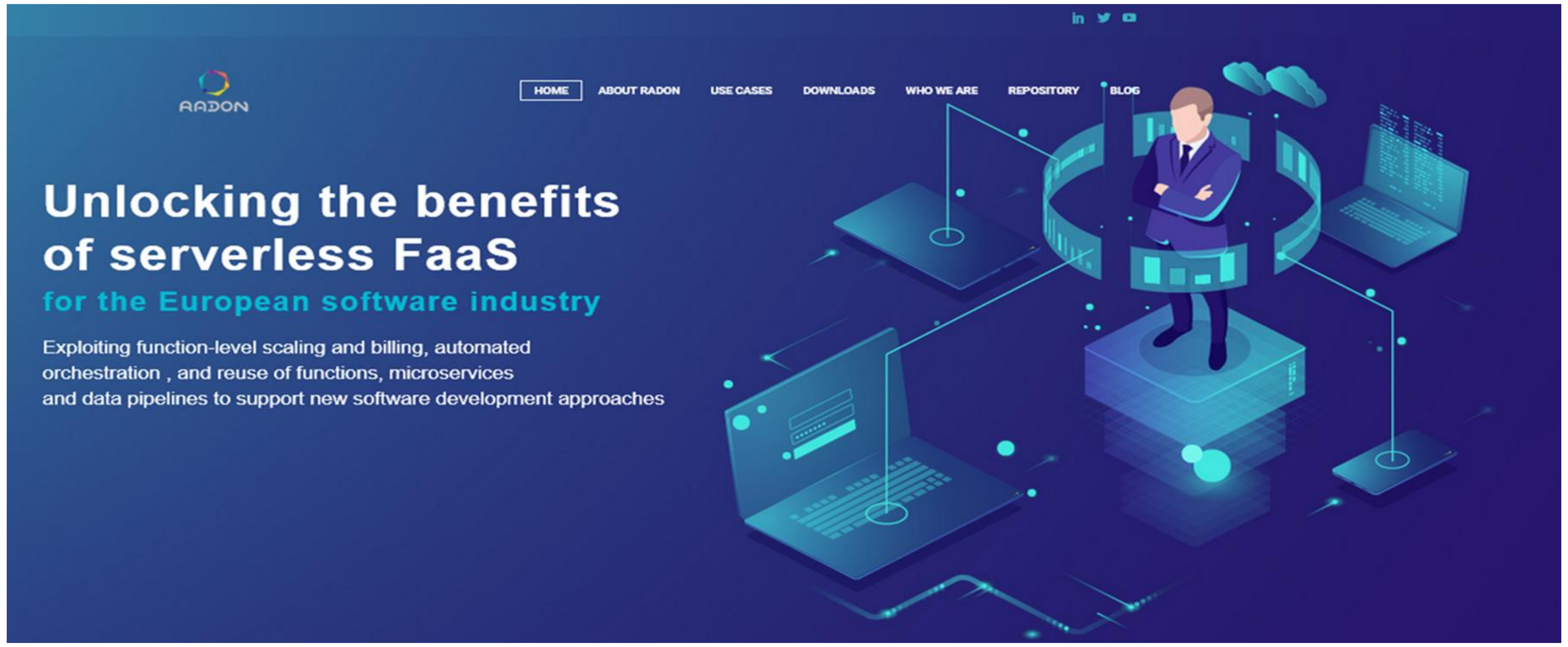
EU http://radon-h2020.eu

Pinned repositories

radon-methodology

Getting started: RADON Methodology for developing microservices and serverless applications

☆ 2 8



Unlocking the benefits of serverless FaaS

for the European software industry

Exploiting function-level scaling and billing, automated orchestration , and reuse of functions, microservices and data pipelines to support new software development approaches

https://github.com/radon-h2020

https://radon-h2020.eu/

Thank you for your attention! ☺