TECH
EXE
TER_

Policy as Code

Why you should, how you can

Olly Stephens, Tech Exeter 2019

TECH EXE TER_

Agenda

- Scene setting
- A quick introduction to Open Policy Agent
- OPA at run time examples
- Shift-left testing
- OPA at build time examples

About Me

- Architect; Technologist; Gopher
- Head of Platform Engineering at Adarga Ltd
- Self-confessed giant shoulder standerer
- *(today's thanks go to Gareth Rushgrove)*

TECH
EXE
TER_

Congratulations

You've worked really hard and you now have a rock solid Infrastructure as Code setup. No more snowflakes; no more point-and-click configuration. You rely on Terraform and Kubernetes manifests to build your entire tech stack. It's done declaratively - you specify what it should look like and the tooling makes it happen. Life is good.

TECH
EXE
TER_

And - somewhere - we have some policies written down

TECH
EXE
TER_

But where are the guard rails?

- Don't do stupid things
- Follow our in-house conventions

TECH
EXE
TER_

Open Policy Agent

A policy enforcement engine for configuration

TECH EXE TER_

Without OPA, you need to implement policy management for your service from scratch. Required components must be carefully designed, implemented, and tested to ensure correct behavior and a positive user experience. That's a lot of work.

OPA includes everything you need in order to policy enable any service.

OPA

Before


Your Service

\+ 
Policy Grammar

\+ 
Policy Compiler

\+ 
Security

\+ 
Interactive Queries

\+ 
Transactions

After


Your Service

\+ 
OPA Integration

= 
Your Service
Policy Enabled

TECH
EXE
TER_

The REGO Language

TECH
EXE
TER_

The REGO Language

- OPA is purpose built for reasoning about information represented in structured documents. The data that your service and its users publish can be inspected and transformed using OPA’s native query language Rego.

TECH
EXE
TER_

The REGO Language

- OPA is purpose built for reasoning about information represented in structured documents. The data that your service and its users publish can be inspected and transformed using OPA’s native query language Rego.
- Rego was inspired by Datalog, which is a well understood, decades old query language. Rego extends Datalog to support structured document models such as JSON.

TECH
EXE
TER_

The REGO Language

- OPA is purpose built for reasoning about information represented in structured documents. The data that your service and its users publish can be inspected and transformed using OPA’s native query language Rego.
- Rego was inspired by Datalog, which is a well understood, decades old query language. Rego extends Datalog to support structured document models such as JSON.
- Rego focuses on providing powerful support for referencing nested documents and ensuring that queries are correct and unambiguous.

TECH
EXE
TER_

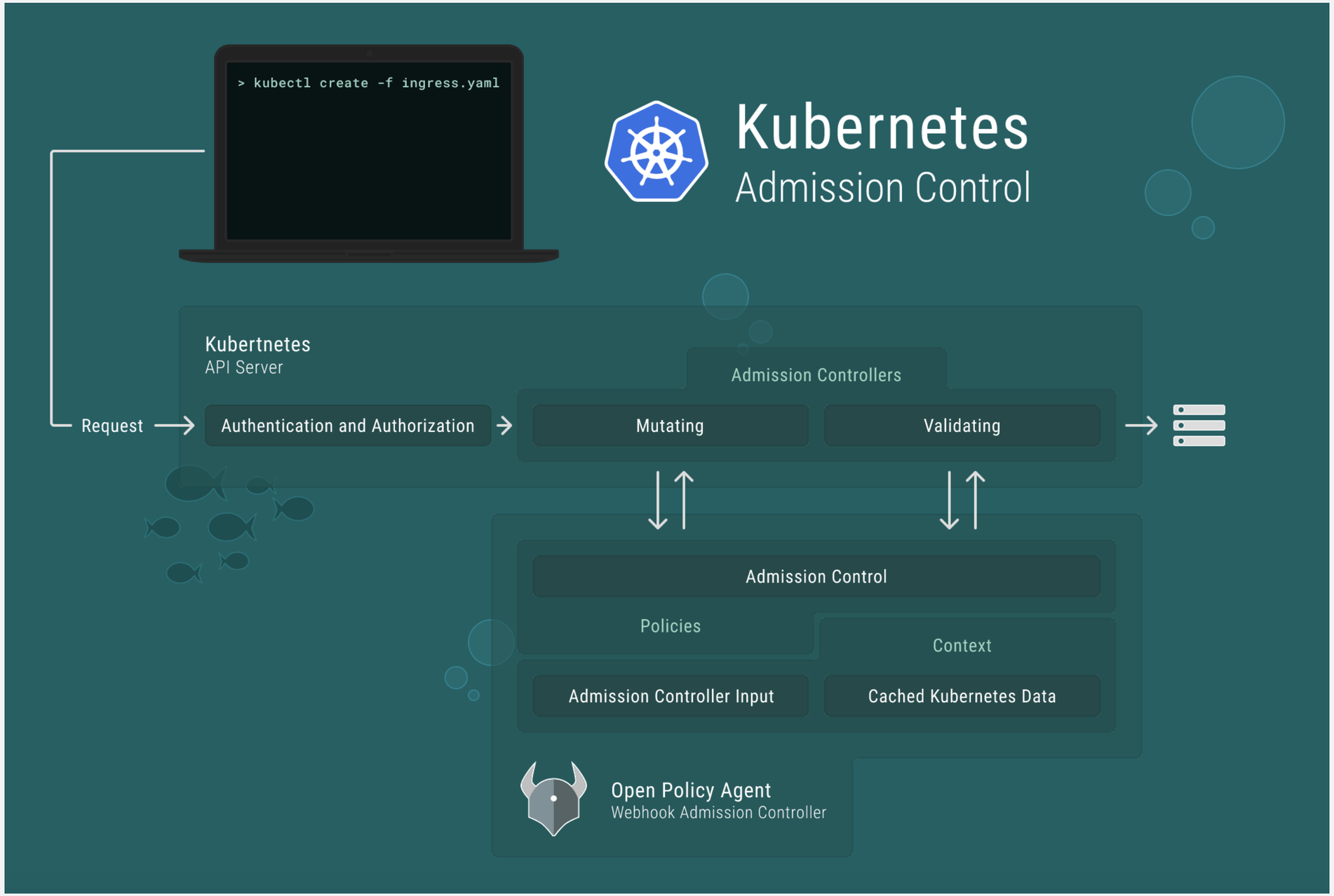
The REGO Language

- OPA is purpose built for reasoning about information represented in structured documents. The data that your service and its users publish can be inspected and transformed using OPA's native query language Rego.
- Rego was inspired by Datalog, which is a well understood, decades old query language. Rego extends Datalog to support structured document models such as JSON.
- Rego focuses on providing powerful support for referencing nested documents and ensuring that queries are correct and unambiguous.
- Rego is declarative so policy authors can focus on what queries should return rather than how queries should be executed. These queries are simpler and more concise than the equivalent in an imperative language.

TECH
EXE
TER_

Kubernetes Example

> Developers are not allowed to create public facing services in the DEV kube cluster


> kubectl create -f ingress.yaml
Kubernetes
Admission Control
Kubertnetes
API Server
Admission Controllers
Request
Authentication and Authorization
Mutating
Validating
Admission Control
Policies
Context
Admission Controller Input
Cached Kubernetes Data
Open Policy Agent
Webhook Admission Controller

TECH
EXE
TER_

```
package kubernetes.admission

operations = {"CREATE", "UPDATE"}

deny[msg] {
    input.request.kind.kind == "Service"
    operations[input.request.operation]
    input.request.object.spec.type == "LoadBalancer"
    input.request.object.annotations[
                "cloud.google.com/load-balancer-type"] != "Internal"
    msg := "Load balancer services must be internal in this cluster"
}
```

```
package kubernetes.admission

operations = {"CREATE", "UPDATE"}

deny[msg] {
    input.request.kind.kind == "Service"
    operations[input.request.operation]
    input.request.object.spec.type == "LoadBalancer"
    input.request.object.annotations[
                "cloud.google.com/load-balancer-type"] != "Internal"
    msg := "Load balancer services must be internal in this cluster"
}
```

If we are creating or updating a Service.

```
package kubernetes.admission

operations = {"CREATE", "UPDATE"}

deny[msg] {
    input.request.kind.kind == "Service"
    operations[input.request.operation]
    input.request.object.spec.type == "LoadBalancer"
    input.request.object.annotations[
                "cloud.google.com/load-balancer-type"] != "Internal"
    msg := "Load balancer services must be internal in this cluster"
}
```

And it's type is LoadBalancer.

```
package kubernetes.admission

operations = {"CREATE", "UPDATE"}

deny[msg] {
    input.request.kind.kind == "Service"
    operations[input.request.operation]
    input.request.object.spec.type == "LoadBalancer"
    input.request.object.annotations[
                "cloud.google.com/load-balancer-type"] ≠ "Internal"
    msg := "Load balancer services must be internal in this cluster"
}
```

And it doesn't have this annotation.

```
package kubernetes.admission

operations = {"CREATE", "UPDATE"}

deny[msg] {
    input.request.kind.kind == "Service"
    operations[input.request.operation]
    input.request.object.spec.type == "LoadBalancer"
    input.request.object.annotations[
                "cloud.google.com/load-balancer-type"] != "Internal"
    msg := "Load balancer services must be internal in this cluster"
}
```

Then deny the request.

TECH
EXE
TER_

Kubernetes Example

> Ingress names must be whitelisted

TECH
EXE
TER_

```
package kubernetes.admission

import data.kubernetes.namespaces

operations = {"CREATE", "UPDATE"}

deny[msg] {
    input.request.kind.kind == "Ingress"
    operations[input.request.operation]
    host := input.request.object.spec.rules[_].host
    not fqdn_matches_any(host, valid_ingress_hosts)
    msg := sprintf("invalid ingress host %q", [host])
}

valid_ingress_hosts = {host |
    whitelist := namespaces[input.request.namespace]
                        .metadata.annotations["ingress-whitelist
    hosts := split(whitelist, ",")
    host := hosts[_]
}
```

TECH
EXE
TER_

```
package kubernetes.admission

import data.kubernetes.namespaces

operations = {"CREATE", "UPDATE"}

deny[msg] {
    input.request.kind.kind == "Ingress"
    operations[input.request.operation]
    host := input.request.object.spec.rules[_].host
    not fqdn_matches_any(host, valid_ingress_hosts)
    msg := sprintf("invalid ingress host %q", [host])
}

valid_ingress_hosts = {host |
    whitelist := namespaces[input.request.namespace]
                    .metadata.annotations["ingress-whitelist
    hosts := split(whitelist, ",")
    host := hosts[_]
}
```

Is the host name whitelisted?

TECH
EXE
TER_

```
package kubernetes.admission

import data.kubernetes.namespaces

operations = {"CREATE", "UPDATE"}

deny[msg] {
    input.request.kind.kind == "Ingress"
    operations[input.request.operation]
    host := input.request.object.spec.rules[_].host
    not fqdn_matches_any(host, valid_ingress_hosts)
    msg := sprintf("invalid ingress host %q", [host])
}

valid_ingress_hosts = {host |
    whitelist := namespaces[input.request.namespace]
                    .metadata.annotations["ingress-whitelist
    hosts := split(whitelist, ",")
    host := hosts[_]
}
```

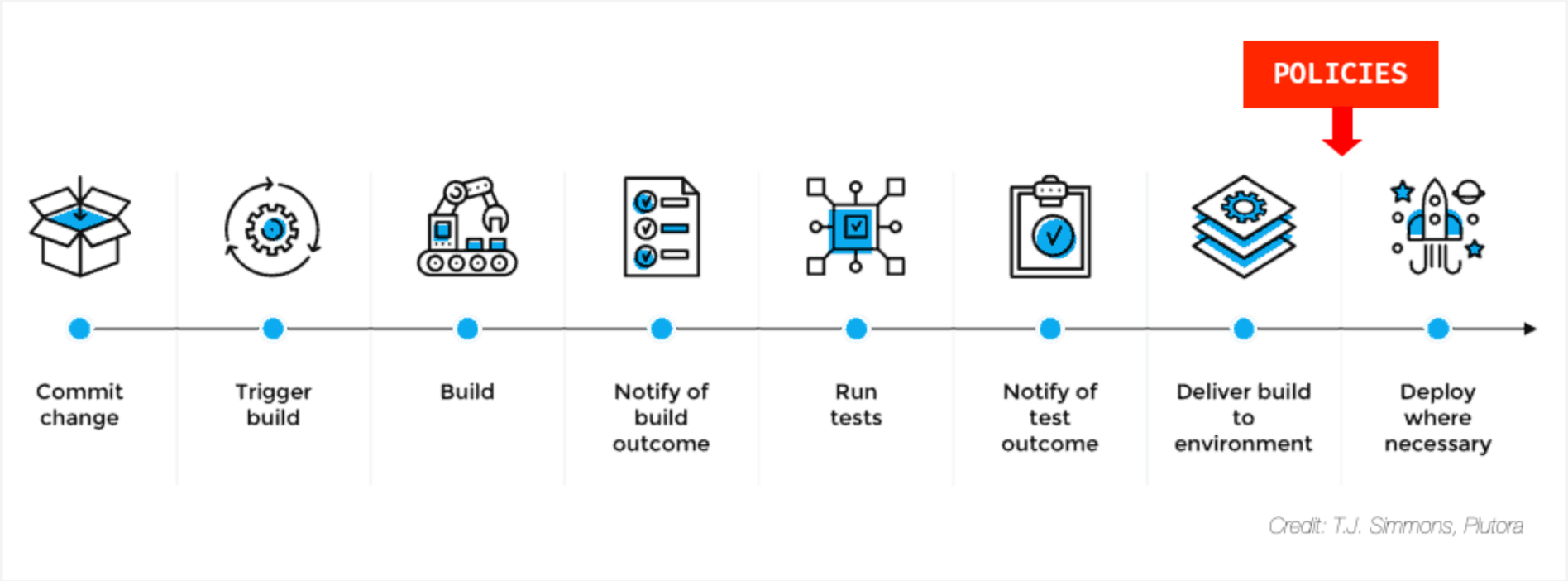
Whitelisted names are attached to namespace

TECH
EXE
TER_

Shift Left

Shift-left testing is an approach to software testing and system testing in which testing is performed earlier in the lifecycle (i.e., moved left on the project timeline). It is the first half of the maxim "Test early and often." Wikipedia

Let's shift left ever so slightly...



TECH
EXE
TER_

conftest

Write tests against structured configuration data using the Open Policy Agent Rego query language.

TECH
EXE
TER_

conftest

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.
- As of today conftest supports:

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.
- As of today conftest supports:
  - YAML

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.
- As of today conftest supports:
  - YAML
  - JSON

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.
- As of today conftest supports:
  - YAML
  - JSON
  - INI

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.
- As of today conftest supports:
  - YAML
  - JSON
  - INI
  - TOML

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.
- As of today conftest supports:
  - YAML
  - JSON
  - INI
  - TOML
  - HCL

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.
- As of today conftest supports:
  - YAML
  - JSON
  - INI
  - TOML
  - HCL
  - CUE

TECH
EXE
TER_

conftest

- conftest allows you to write policies using Open Policy Agent/rego and apply them to one or more configuration files.
- As of today conftest supports:
  - YAML
  - JSON
  - INI
  - TOML
  - HCL
  - CUE
  - Dockerfile

TECH
EXE
TER_

Can still check our kubernetes manifests

```
package main

deny[msg] {
  input.kind = "Deployment"
  not input.spec.template.spec.securityContext.runAsNonRoot = true
  msg = sprintf("Containers in deployment '%s' must not run as root",
                [input.metadata.name])
}

deny[msg] {
  input.kind = "CronJob"
  not input.spec.jobTemplate.spec.template.spec.securityContext.runAsNonRoot = true
  msg = sprintf("Containers in cronjob '%s' must not run as root",
                [input.metadata.name])
}
```

TECH
EXE
TER_

and...

```
package main

labels {
    input.metadata.labels["app.kubernetes.io/name"]
    input.metadata.labels["app.kubernetes.io/instance"]
    input.metadata.labels["app.kubernetes.io/version"]
    input.metadata.labels["app.kubernetes.io/component"]
    input.metadata.labels["app.kubernetes.io/part-of"]
    input.metadata.labels["app.kubernetes.io/managed-by"]
}

deny[msg] {
  input.kind = "Deployment"
  not labels
  msg = sprintf("Deployment '%s' must include standard labels",
                [input.metadata.name])
}
```

TECH
EXE
TER_

but this time, we do it before we push

(as part of a test for a continuous deployment trigger)

```
% conftest test ./bad-deploy.yaml && echo OK
FAIL - ./bad-deploy.yaml - Deployment 'tokenizer' must include standard labels
FAIL - ./bad-deploy.yaml - Containers in deployment 'tokenizer' must not run as root

% conftest test ./good-deploy.yaml && echo OK
OK
```

TECH
EXE
TER_

Terraform Example

“All AWS assets (that support it) must have a `cost_code` tag set.”

```
package main

resources_that_take_tags = {
  "aws_iam_role",
  "aws_instance",
  "aws_internet_gateway",
  "aws_security_group",
  "aws_subnet",
  "aws_vpc"
}

deny[msg] {
  some i
  name := input.resource_changes[i].name
  type := input.resource_changes[i].type
  resources_that_take_tags[type]
  not input.resource_changes[i].change.after.tags.cost_code

  msg := sprintf("%s.%s does not have cost_code tag",
           [type, name])
}
```

```
package main

resources_that_take_tags = {
  "aws_iam_role",
  "aws_instance",
  "aws_internet_gateway",
  "aws_security_group",
  "aws_subnet",
  "aws_vpc"
}

deny[msg] {
  some i
  name := input.resource_changes[i].name
  type := input.resource_changes[i].type
  resources_that_take_tags[type]
  not input.resource_changes[i].change.after.tags.cost_code

  msg := sprintf("%s.%s does not have cost_code tag",
           [type, name])
}
```

There exists some value of i

```
package main

resources_that_take_tags = {
  "aws_iam_role",
  "aws_instance",
  "aws_internet_gateway",
  "aws_security_group",
  "aws_subnet",
  "aws_vpc"
}

deny[msg] {
  some i
  name := input.resource_changes[i].name
  type := input.resource_changes[i].type
  resources_that_take_tags[type]
  not input.resource_changes[i].change.after.tags.cost_code

  msg := sprintf("%s.%s does not have cost_code tag",
           [type, name])
}
```

that is an instance of a resource that supports tags

```
package main

resources_that_take_tags = {
  "aws_iam_role",
  "aws_instance",
  "aws_internet_gateway",
  "aws_security_group",
  "aws_subnet",
  "aws_vpc"
}

deny[msg] {
  some i
  name := input.resource_changes[i].name
  type := input.resource_changes[i].type
  resources_that_take_tags[type]
  not input.resource_changes[i].change.after.tags.cost_code

  msg := sprintf("%s.%s does not have cost_code tag",
           [type, name])
}
```

that does not have cost_code in the set of tags after apply

TECH
EXE
TER_

...with a little hoop jumping

```
% terraform plan -out tfplan
% terraform show -json tfplan | conftest test -
FAIL - aws_internet_gateway.my-vpc-igw does not have cost_code tag
FAIL - aws_vpc.my-vpc does not have cost_code tag
```

TECH
EXE
TER_

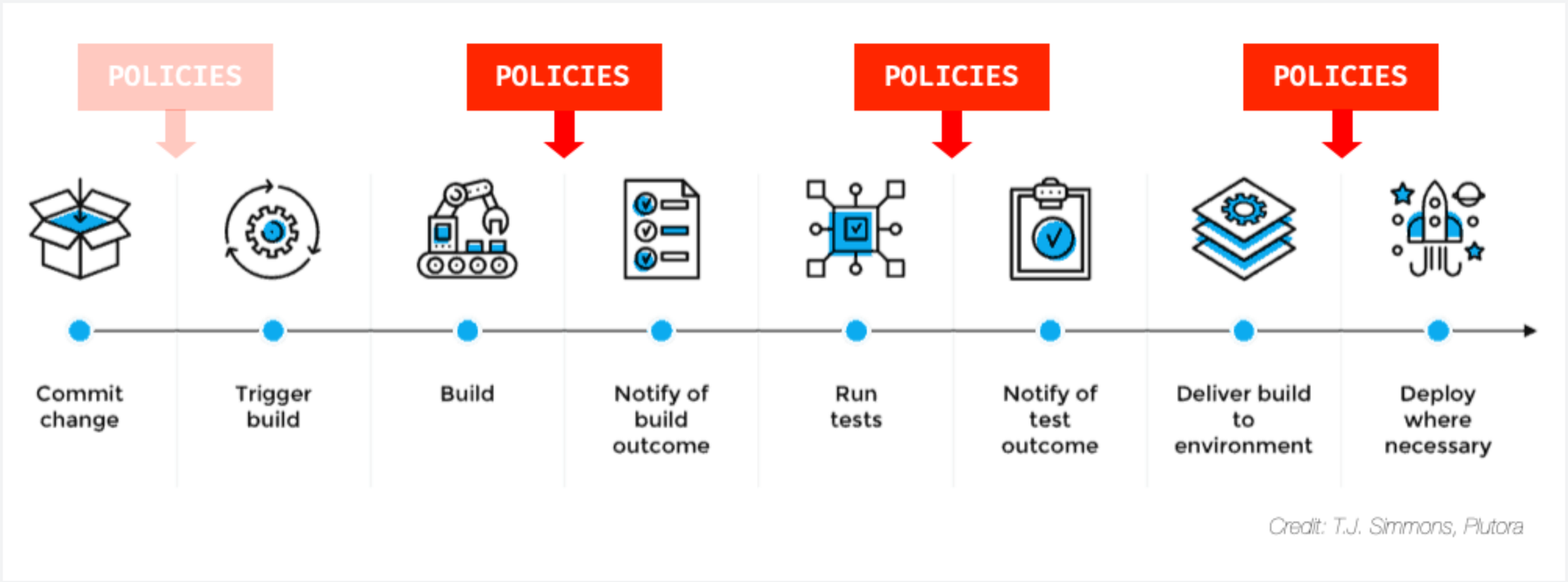
More Terraform

> Control blast radius to protect against catastrophic deployments

```
blast_radius = 10

weights = {
    "aws_autoscaling_group": {"delete": 100, "create": 10, "modify": 1},
    "aws_instance": {"delete": 10, "create": 1, "modify": 1}
}

resource_types = {"aws_autoscaling_group", "aws_instance"}

deny[msg] {
    score > blast_radius
    msg = sprintf("Makes too many changes, scoring %v which
            is greater than current maximum %v", [score, blast_radius])
}

score = s {
    all := [ x |
            some resource_type
            crud := weights[resource_type];
            del := crud["delete"] * num_deletes[resource_type];
            new := crud["create"] * num_creates[resource_type];
            mod := crud["modify"] * num_modifies[resource_type];
```

TECH EXE TER_

Moving back from CD to CI



TECH
EXE
TER_

Snyk

(vulnerability scanning)

> All high severity vulnerabilities left in code must be formally waived by CISO

TECH
EXE
TER_

Snyk policy file

```
# Snyk (https://snyk.io) policy file, patches or ignores known vulnerabilities.
version: v1.13.5
# ignores vulnerabilities until expiry date; change duration by modifying expiry date
ignore:
  SNYK-JAVA-CHQOSLOGBACK-173711:
    - '*':
        reason: pending code refactor (BE-4152)
        expires: 2019-10-10T09:37:38.999Z
  SNYK-JAVA-CHQOSLOGBACK-31407:
    - '*':
        reason: pending code refactor (BE-4152)
        expires: 2019-10-10T09:38:01.367Z
```

--

Synk policy file policy

```
package main

valid_waivers = {
  "SNYK-JAVA-CHQOSLOGBACK-173711",
  "SNYK-JAVA-COMFASTERXMLJACKSONCORE-450207"
}

deny[msg] {
  ignored := { name | input.ignore[name] = _ }
  invalid_waivers := ignored - valid_waivers
  count(invalid_waivers) > 0
  msg = sprintf("Vulnerabilities %s are not in the allowed waiver list",
          [concat(", ", invalid_waivers)])
}
```

TECH
EXE
TER_

Service Config

(grafana.ini example)

grafana.ini

```
app_mode = production
instance_name = ${HOSTNAME}
[server]
protocol = http
http_port = 3000
[users]
allow_sign_up = false
allow_org_create = false
auto_assign_org = true
auto_assign_org_id = 1
auto_assign_org_role = Viewer
verify_email_enabled = false
```

--

Policy file

```
package main

deny[msg] {
  not input.alerting.enabled = "true"
  msg = "Alerting should turned on"
}

deny[msg] {
  not input.server.http_port = "3443"
  msg = "Grafana port should be 3443"
}

deny[msg] {
  not input.server.protocol = "https"
  msg = "Grafana should use default https"
}
```

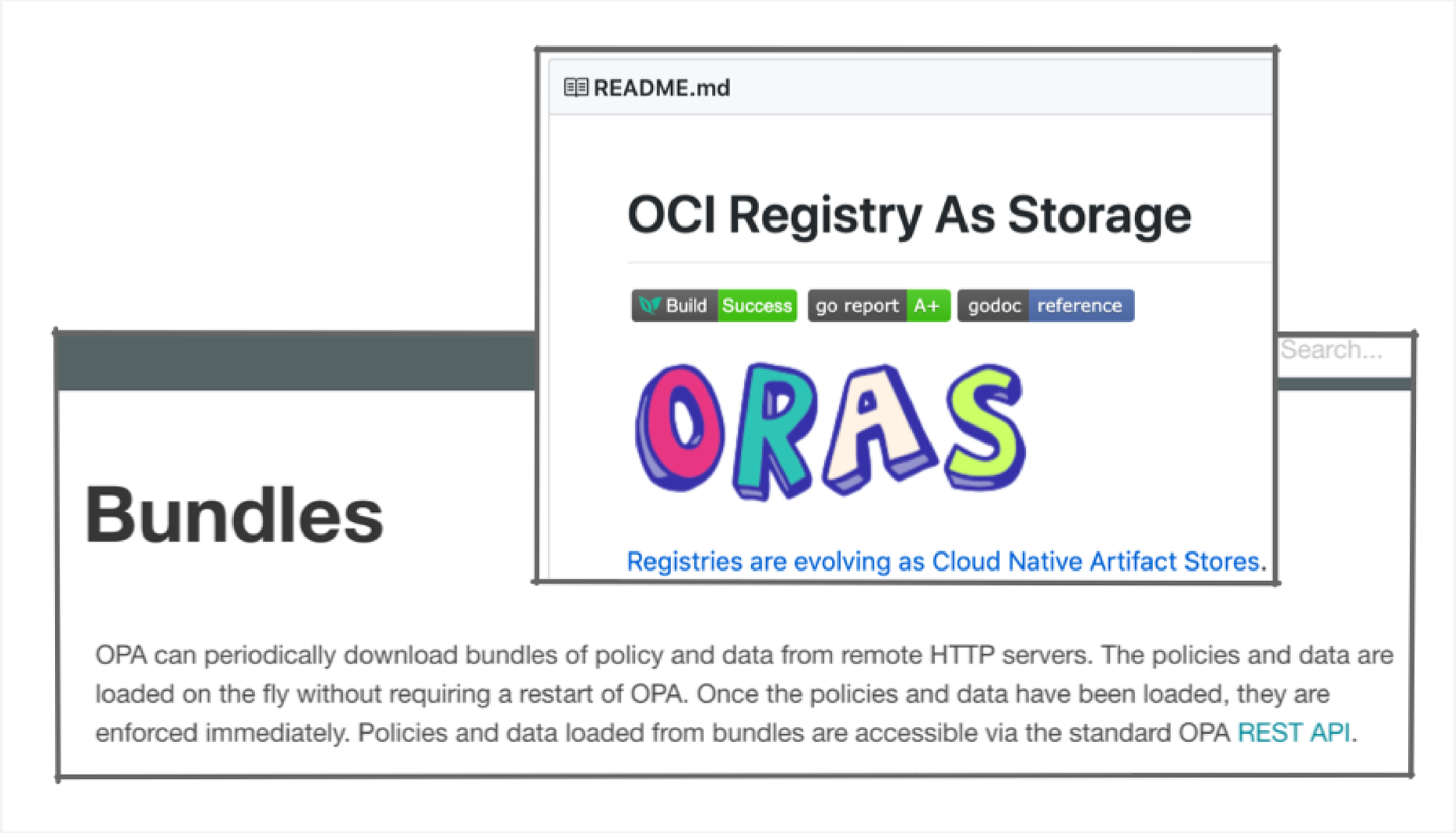
```
% conftest test grafana.ini
FAIL - grafana.ini - Alerting should turned on
FAIL - grafana.ini - Grafana port should be 3443
FAIL - grafana.ini - Grafana should use default https
FAIL - grafana.ini - Users should verify their e-mail address
```

TECH
EXE
TER_

Registry of policies

(division of responsibility)

Leveraging the ecosystem



```
% ls
waivers.snyk

% docker login xxx.azurecr.io
% conftest pull xxx.azurecr.io/policies/snyk:latest
% ls
waivers.snyk   policy/

% conftest test waivers.snyk
FAIL - waivers.snyk - Vulnerabilities SNYK-JAVA-CHQOSLOGBACK-31407,
SNYK-JAVA-COMFASTERXMLJACKSONCORE-174736, SNYK-JAVA-COMFASTERXMLJACKSONCORE-31507,
SNYK-JAVA-COMFASTERXMLJACKSONCORE-31573, SNYK-JAVA-COMFASTERXMLJACKSONCORE-72884,
SNYK-JAVA-COMFASTERXMLJACKSONCORE-32043, SNYK-JAVA-COMFASTERXMLJACKSONCORE-32044,
SNYK-JAVA-COMFASTERXMLJACKSONCORE-32111 are not in the allowed waiver list
```

TECH
EXE
TER_

Testing policies

“All code repositories should contain unit tests.”

“Only code built by official CI/CD processes can be used in production.”

```
package main

empty(value) {
  count(value) == 0
}
no_violations {
  empty(deny)
}

test_deployment_without_security_context {
  deny["Containers in deployment 'test' must not run as root"]
    with input as {"kind": "Deployment", "metadata": {"name": "test"}}
}

test_deployment_with_security_context {
  no_violations with input as {
    "kind": "Deployment",
    "metadata": { "name": "test" },
    "spec": { "template": { "spec": { "securityContext": { "runAsNonRoot": true }}}}
  }
}
```

TECH
EXE
TER_

Using OPA's built-in test harness

```
% opa test --verbose policy/run-as-non-root.rego policy/run-as-non-root-tests.rego
data.main.test_deployment_without_security_context: PASS (2.2231ms)
data.main.test_deployment_with_security_context: PASS (450.375µs)

PASS: 2/2
```

TECH
EXE
TER_

Summing Up

- Policies should be coded, not scribed
  - CI/CD pipelines need conformance checks
- OPA is a powerful and appropriate framework
  - *(side note: we should use it in our services)*
- `conftest` allows us to leverage OPA in the places we need
  - *shift left - check early, check often*
- Registry support allows us to clearly deliniate responsibilities
- Ecosystem is immature, but worth investment

TECH
EXE
TER_

Questions?

olly@marste.net