CI tools

Jenkins

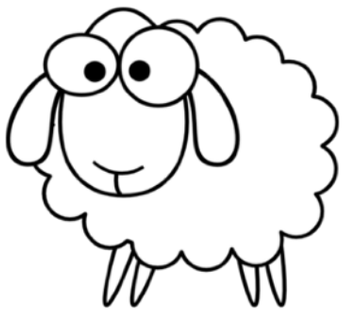
Travis

codeship

TeamCity

AppVeyor

Circle CI



LambCI

AWS CodePipeline

Concourse CI

design goals

design goals

- “pipeline as code”

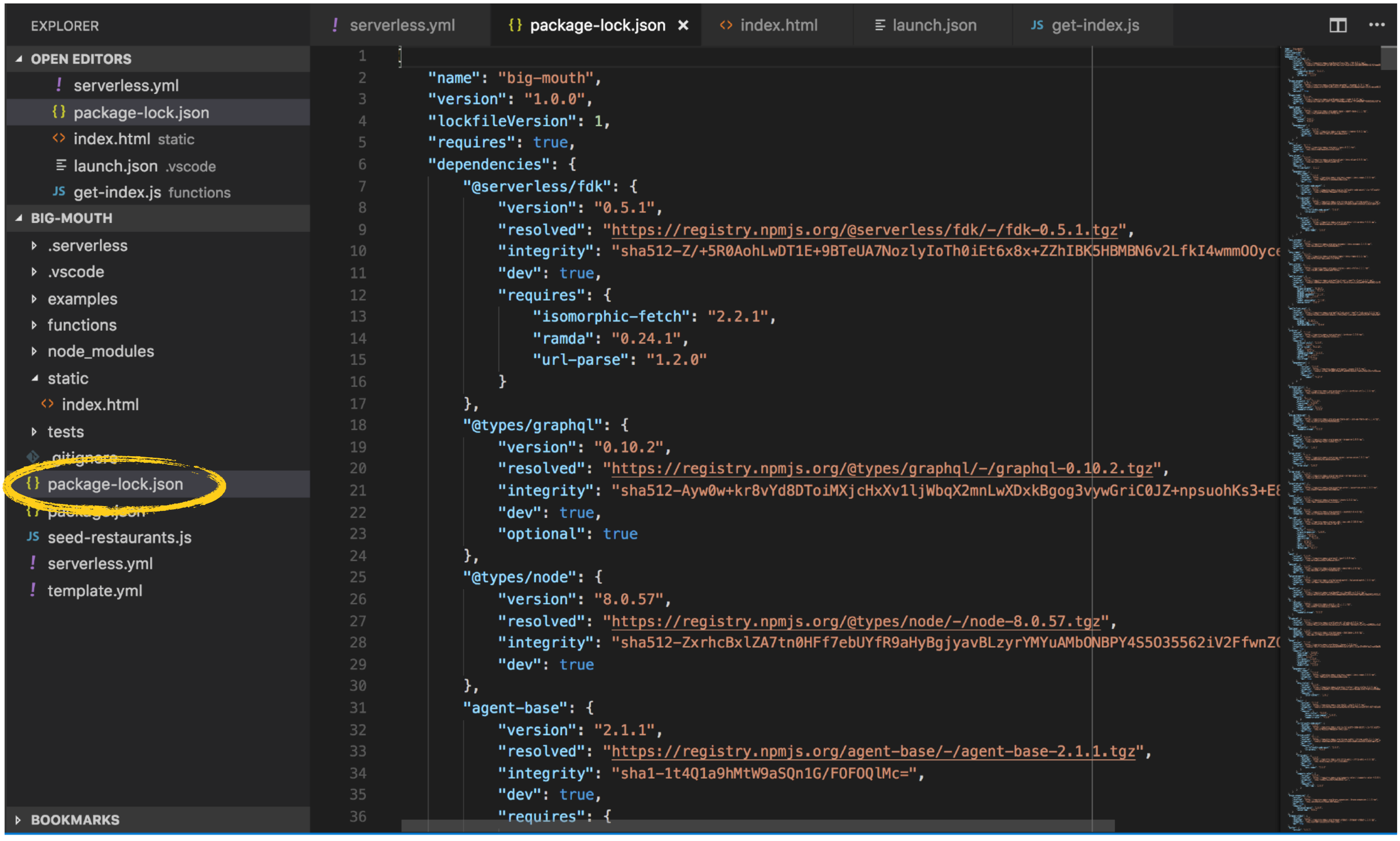
design goals

- “pipeline as code”
- reproducible builds

reproducible builds

NPM default - get latest
“compatible” version, ie. 1.X.X

reproducible builds



use `npm ci` during CI to restore exact version

Postmortem for Malicious Packages Published on July 12th, 2018

Summary

On July 12th, 2018, an attacker compromised the npm account of an ESLint maintainer and published malicious versions of the `eslint-scope` and `eslint-config-eslint` packages to the npm registry. On installation, the malicious packages downloaded and executed code from `pastebin.com` which sent the contents of the user's `.npmrc` file to the attacker. An `.npmrc` file typically contains access tokens for publishing to npm.

The malicious package versions are `eslint-scope@3.7.2` and `eslint-config-eslint@5.0.2`, both of which have been unpublished from npm. The `pastebin.com` paste linked in these packages has also been taken down.

npm has revoked all access tokens issued before 2018-07-12 12:30 UTC. As a result, all access tokens compromised by this attack should no longer be usable.

The maintainer whose account was compromised had reused their npm password on several other sites and did not have two-factor authentication enabled on their npm account.

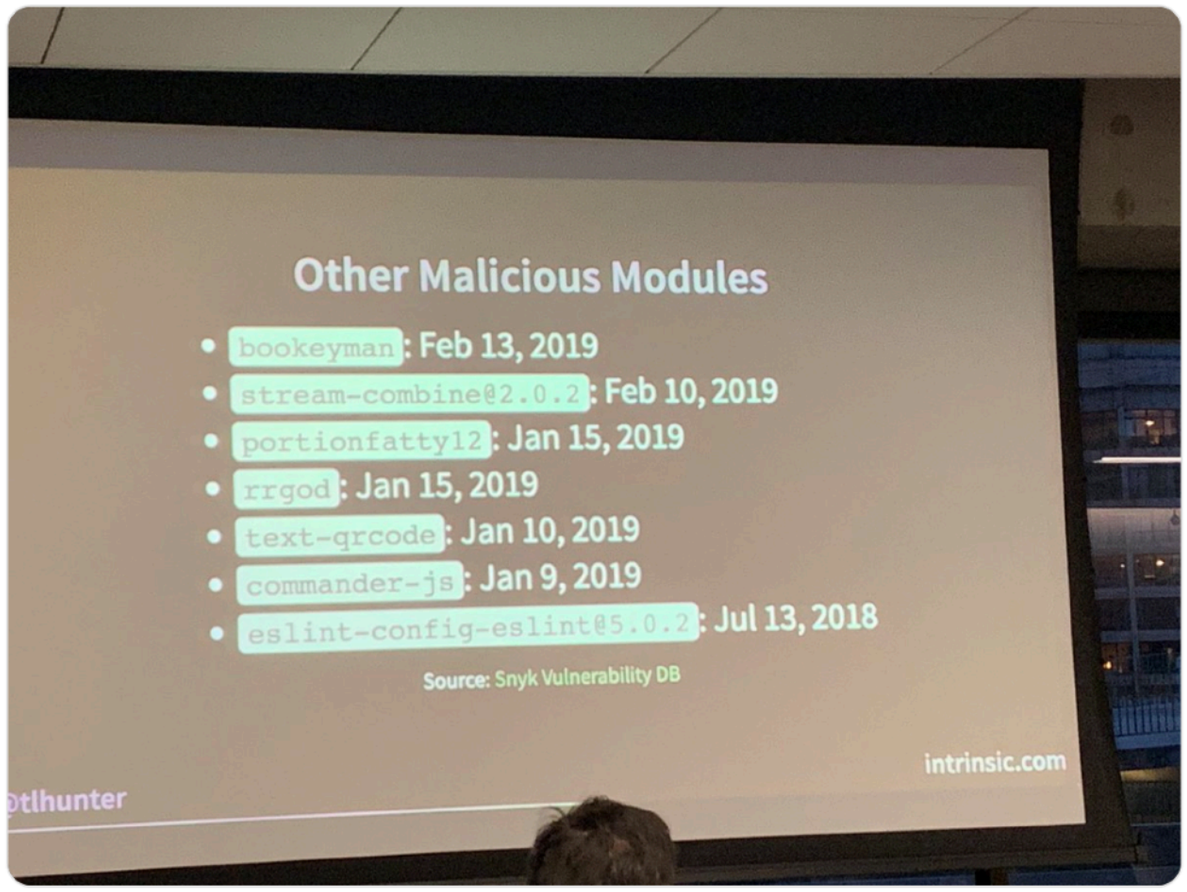
> We, the ESLint team, are sorry for allowing this to happen. We hope that other package maintainers can learn from our mistakes and improve the security of the whole npm ecosystem.

https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes

Aleksandar Simovic
@simalexan

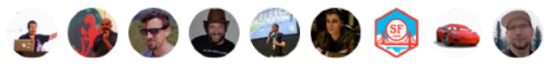
Following

Interesting talk by @tlhunter
“Real World Attacks in the npm Ecosystem”
#SFNode meetup



3:55 AM - 5 Apr 2019 from Capital One

2 Retweets **8** Likes



2 8

brian ɟeɹoux 💎
@brianleroux

Following

🤔 A few things we npm package authors/maintainers can do today:

- 🔒 Enable 2fa for everyone like rn
- 🔑 Revoke/reset all your tokens *
- 📌 Pin all deps to an exact version in package.json as a practice so auto-upgrades cannot burn us

* Don't forget your CI! 😭

6:00 PM - 12 Jul 2018

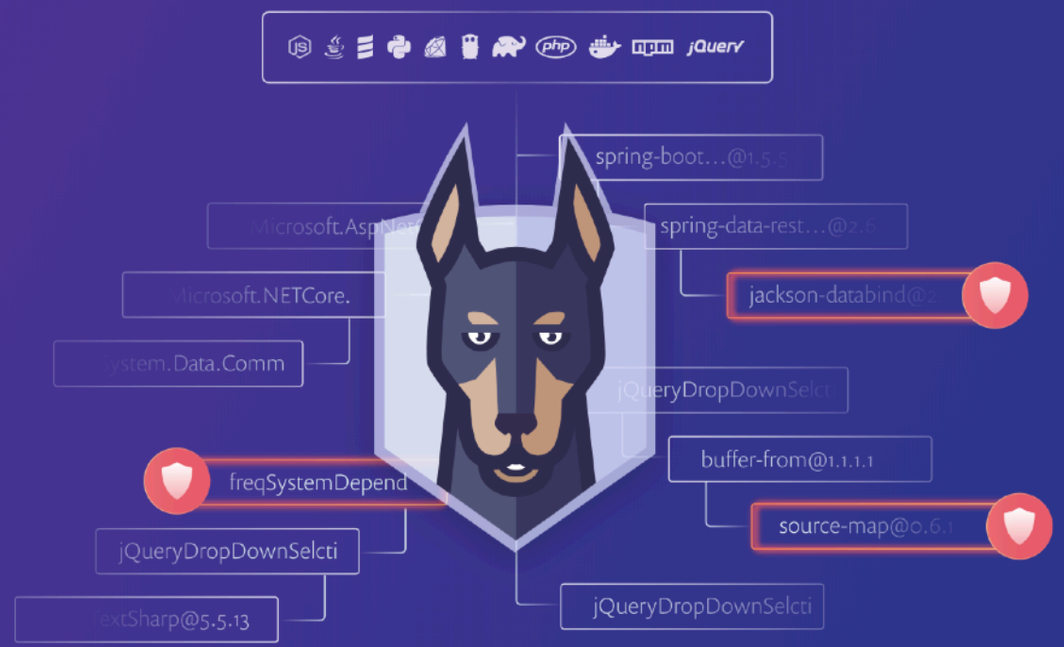
54 Retweets **108** Likes

7 54 108

Develop fast.
Stay secure.

Enabling more than 400,000 developers to continuously find and fix vulnerabilities in open source libraries and containers.

SIGN UP FOR FREE



PROTECTED BY SNYK

Google mongoDB Skyscanner mastercard. salesforce intuit New Relic. BBC ASOS discover fashion online Telstra

---

Why is **Snyk** different?

design goals

- “pipeline as code”
- reproducible builds
- fast!

prefer CI tools that lets you use containers to run each step

build script

```
if [ "$1" = "deploy" ] && [ $# -eq 4 ]; then
  STAGE=$2
  REGION=$3
  PROFILE=$4

  npm install
  AWS_PROFILE=$PROFILE 'node_modules/.bin/sls' deploy -s $STAGE -r $REGION
elif [ "$1" = "int-test" ] && [ $# -eq 4 ]; then
  STAGE=$2
  REGION=$3
  PROFILE=$4

  npm install
  AWS_PROFILE=$PROFILE npm run int-$STAGE
elif [ "$1" = "acceptance-test" ] && [ $# -eq 4 ]; then
  STAGE=$2
  REGION=$3
  PROFILE=$4

  npm install
  AWS_PROFILE=$PROFILE npm run acceptance-$STAGE
else
  usage
  exit 1
fi
```

build script

Build

Execute shell

Command

```
./build.sh test test eu-west-1 non_prod
```

See the list of available environment variables

Execute shell

Command

```
./build.sh deploy test eu-west-1 non_prod
```

See the list of available environment variables

Execute shell

Command

```
./build.sh validate test eu-west-1 non_prod
```

See the list of available environment variables

Why you should use temporary stacks when you do serverless

AWS, CloudFormation, Programming, Serverless / September 12, 2019

*Check out my new course **Learn you some Lambda best practice for great good!** and learn the best practices for performance, cost, security, resilience, observability and scalability.*

One of the benefits of serverless is the pay-per-use pricing model you get from the platform. That is, if your code doesn't run, you don't pay for them!

Combined with the simplified deployment flow (compared with applications running in containers or VMs) it has enabled many teams to make use of temporary CloudFormation stacks.

In this post, let's talk about two ways you should use temporary CloudFormation stacks, and why. ***Disclaimer**: this shouldn't be taken as a prescription. It's a general approach that has pros and cons, which we will discuss along the way.*

Temporary stacks for feature branches

It's common for teams to have multiple AWS accounts, one for each environment. While there doesn't seem to be a consensus on how to use these environments, I tend to follow these conventions:

- `dev` is shared by the team, this is where the latest development changes are deployed and tested end-to-end. This environment is unstable by nature, and shouldn't be used by other teams.
- `test` is where other teams can integrate with your team's work. This environment should be fairly stable so to not slow down other teams.

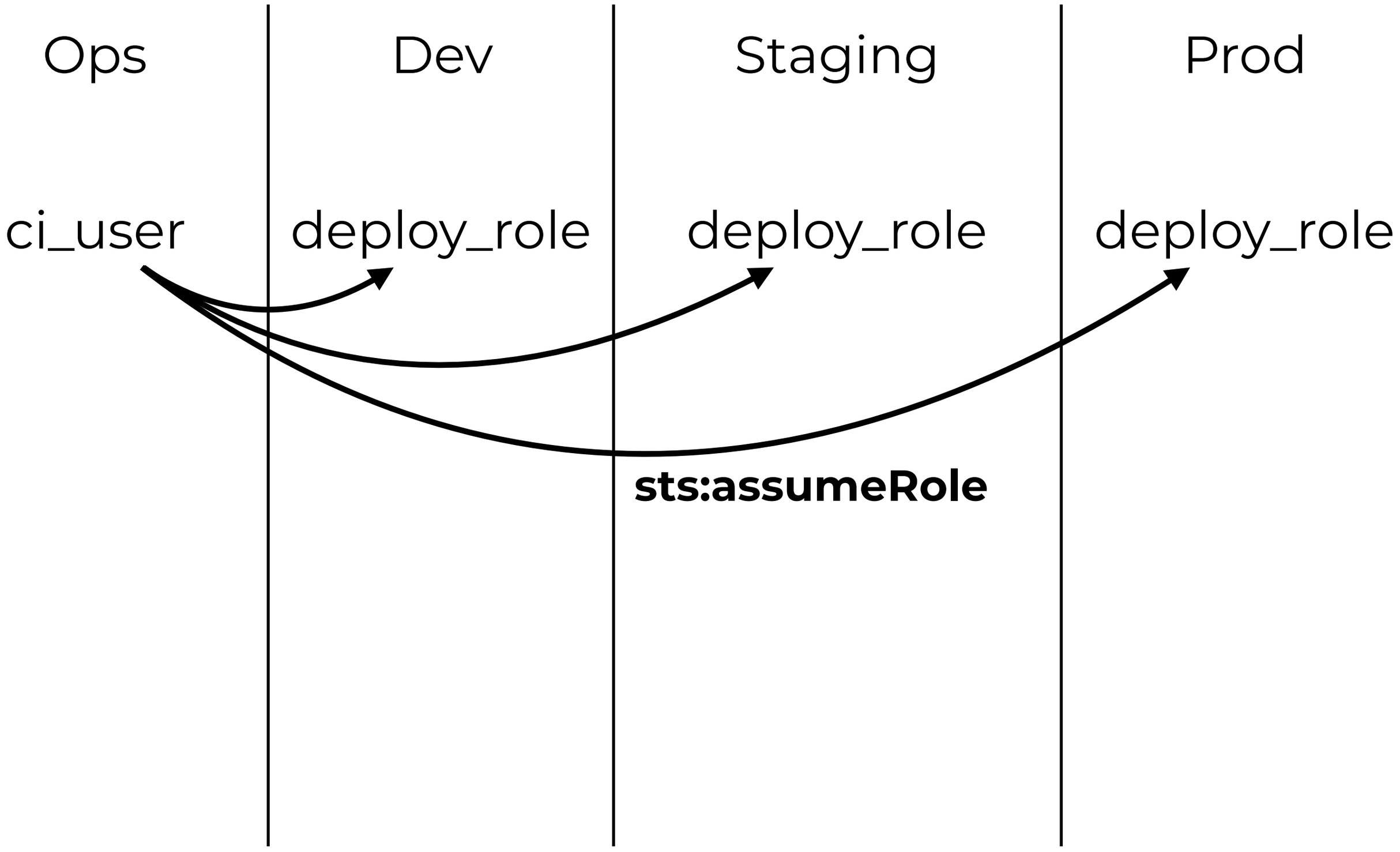
https://theburningmonk.com/2019/09/why-you-should-use-temporary-stacks-when-you-do-serverless/

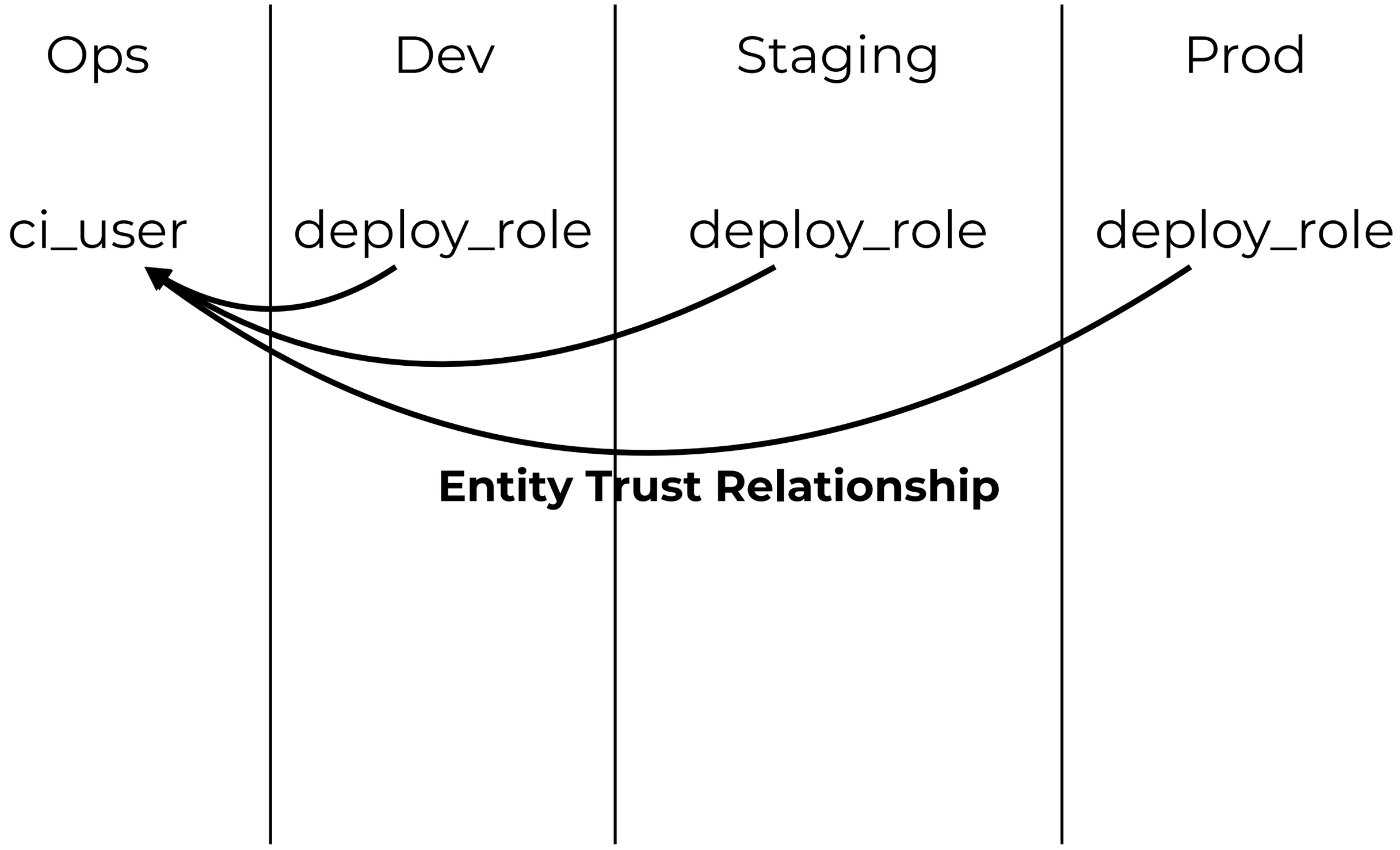
design goals

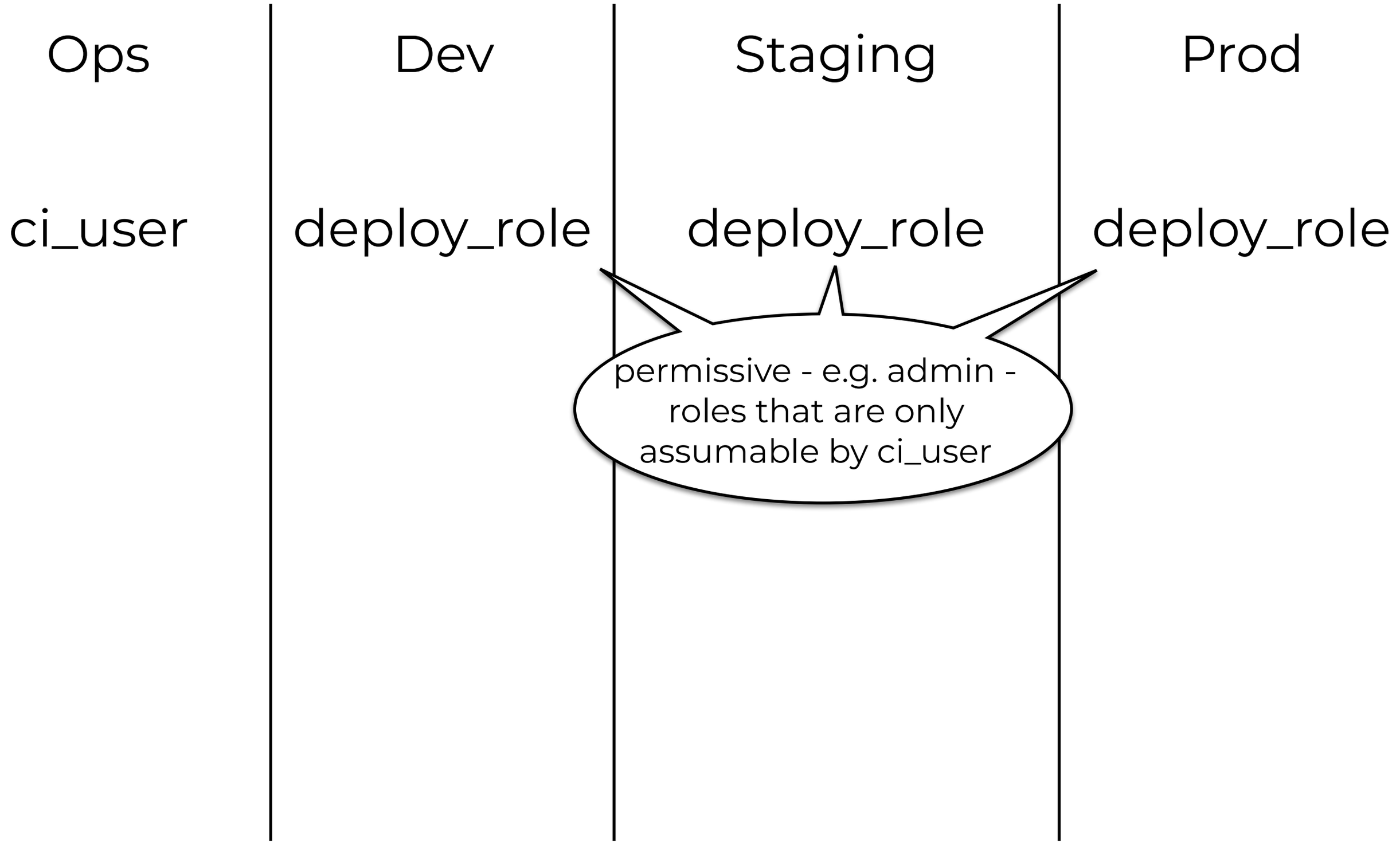
- “pipeline as code”
- reproducible builds
- fast!
- secure

Ops	Dev	Staging	Prod


Ops
Dev
Staging
Prod
ci_user
NO local permissions


Ops
Dev
Staging
Prod
ci_user
deploy_role
deploy_role
deploy_role
sts:assumeRole


Ops
Dev
Staging
Prod
ci_user
deploy_role
deploy_role
deploy_role
Entity Trust Relationship

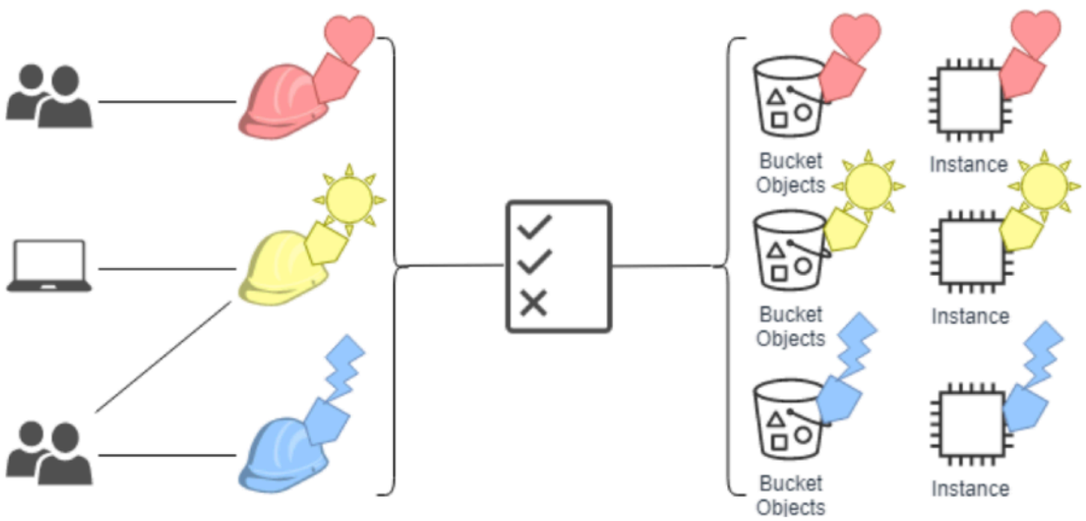

Ops
Dev
Staging
Prod
ci_user
deploy_role
deploy_role
deploy_role
permissive - e.g. admin -
roles that are only
assumable by ci_user

What Is ABAC for AWS?

PDF | Kindle | RSS

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. Tags can be attached to IAM principals (users or roles) and to AWS resources. You can create a single ABAC policy or small set of policies for your IAM principals. These ABAC policies can be designed to allow operations when the principal's tag matches the resource tag. ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

For example, you can create three roles with the `access-project` tag key. Set the tag value of the first role to `Heart`, the second to `Sun`, and the third to `Lightning`. You can then use a single policy that allows access when the role and the resource are tagged with the same value for `access-project`. For a detailed tutorial that demonstrates how to use ABAC in AWS, see Tutorial: Using Tags for Attribute-Based Access Control in AWS.



https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_attribute-based-access-control.html

Ops	Dev	Staging	Prod
ci_user	service1_role	service1_role	service1_role
	service2_role	service2_role	service2_role

use ABAC to lock down resources that can be deployed