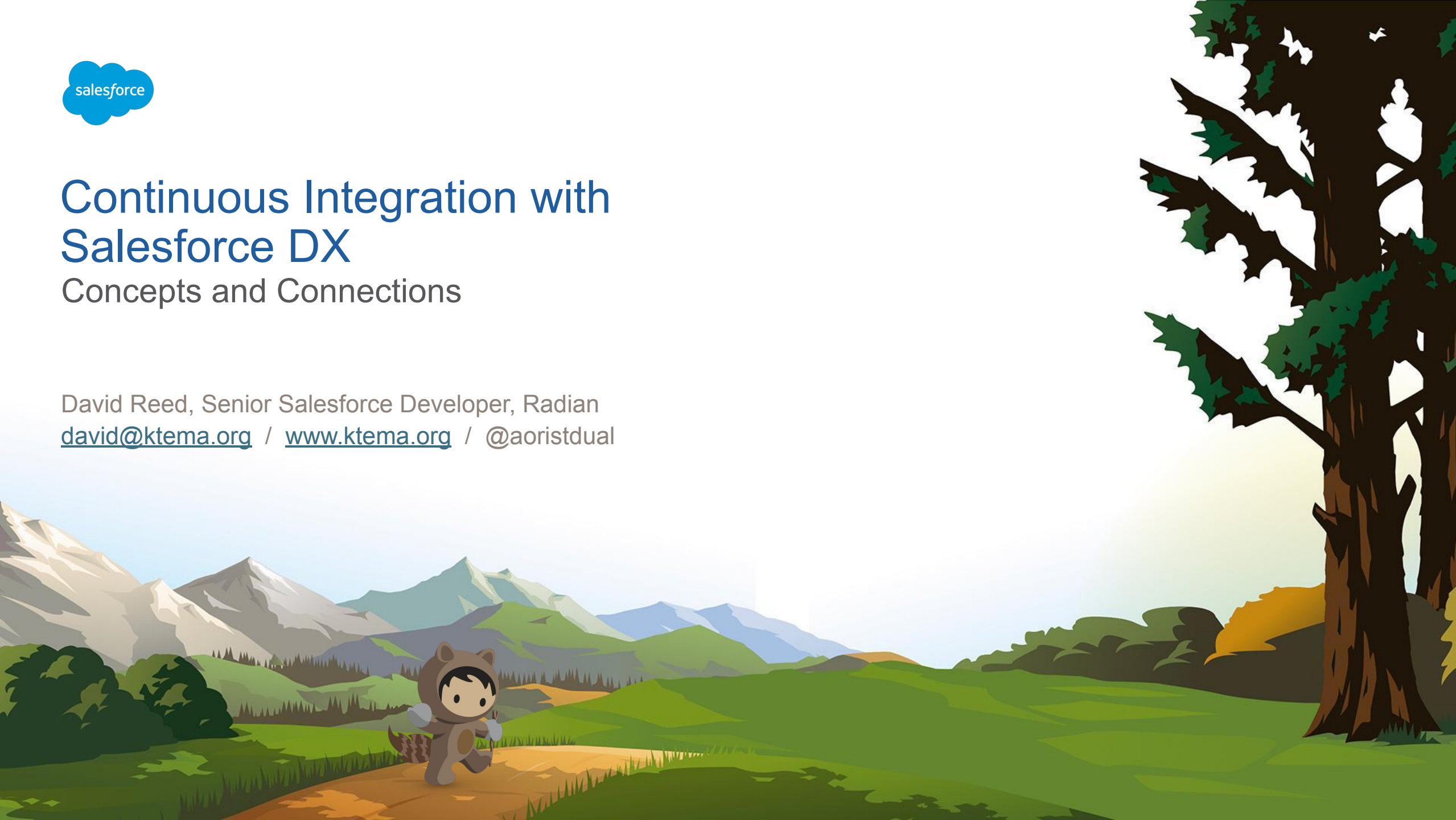


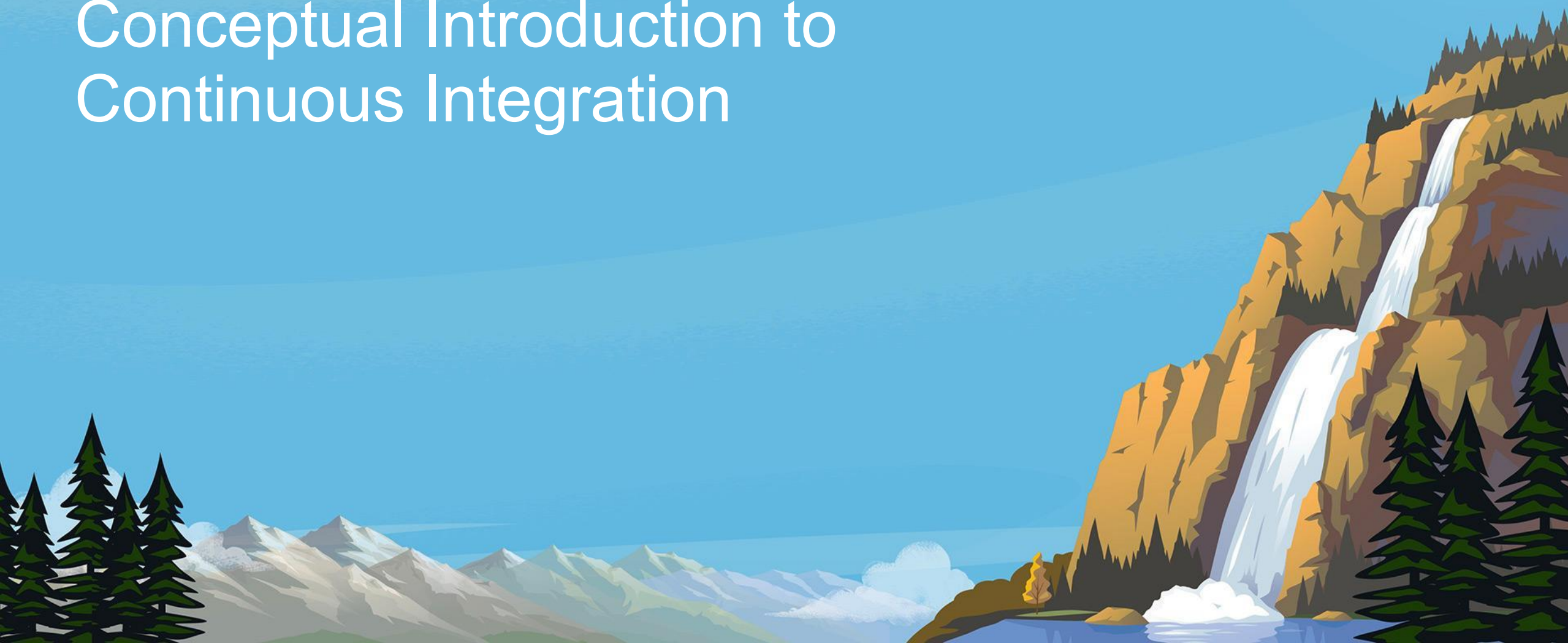
# Continuous Integration with Salesforce DX

## Concepts and Connections

David Reed, Senior Salesforce Developer, Radian
david@ktema.org / www.ktema.org / @aoristdual


David Reed
Senior Salesforce Developer,
Radian
david@ktema.org / @aoristdual

# Conceptual Introduction to Continuous Integration



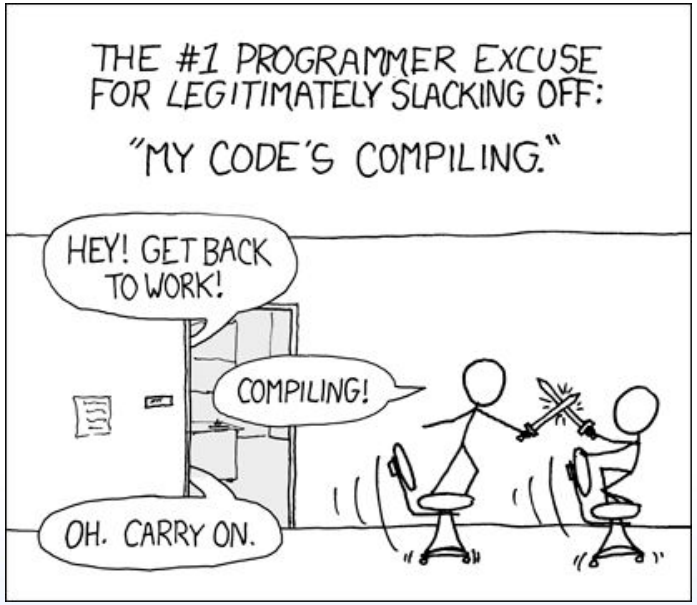
# What Is Continuous Integration?

Methodology for building and testing software **constantly** and **automatically**

Use scripting to reduce or eliminate human intervention

Focus on **consistency**, **repeatability**

Add tools to increase insight and value




Your code is never **not** compiling.

# Continuous integration is for…

Developers who write code

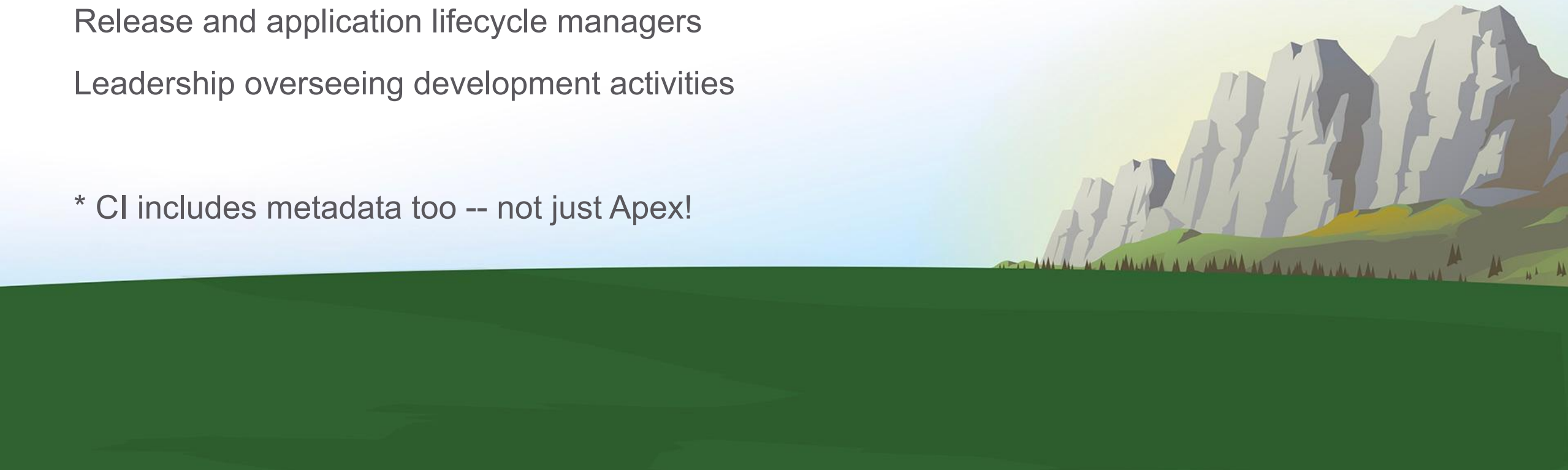
Admins and app builders who work in an org that has code

Implementors and solution architects in code-based or code/declarative orgs

Release and application lifecycle managers

Leadership overseeing development activities

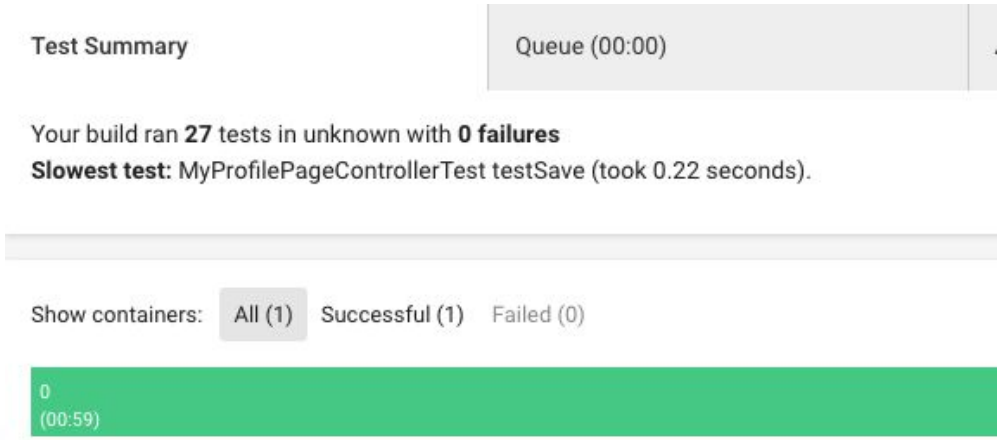
* CI includes metadata too -- not just Apex!



# CI Flow: User Perspective

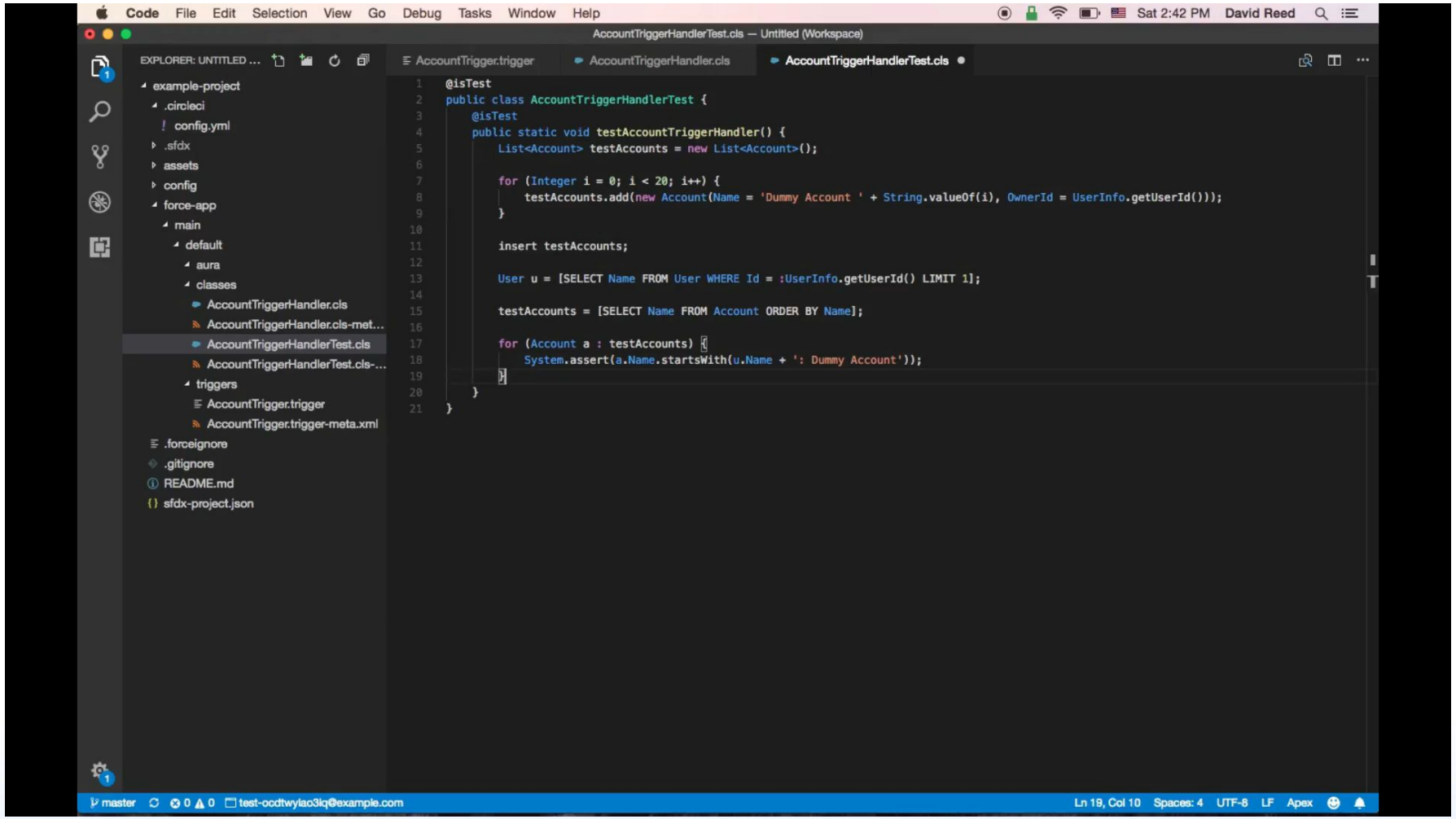
Write Code → Push → Get Feedback

```
$ git push
```



# CI Flow: User Perspective

## Finding and fixing a regression in under ~~two~~ ten minutes with CI



# The Value Proposition

## CI in the Development Lifecycle

# Visibility

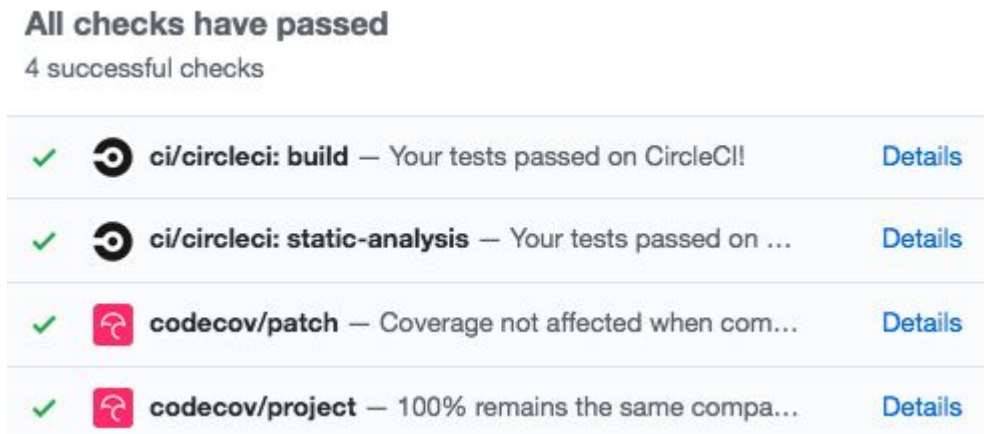
## Across your codebase, every commit

Tests passing? ✓

Regressions? ✓

Overall test coverage? ✓

Coding best practices? ✓



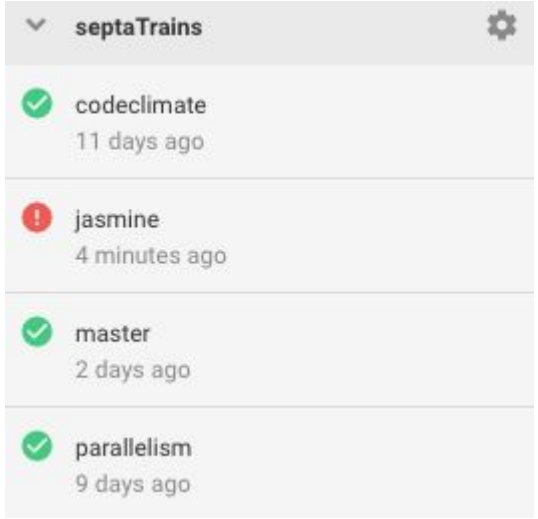
# Risk Reduction

Never unknowingly merge broken code.

Find more regressions faster - no waiting on manual test cycles.

Surface areas of risk.

Iterate and branch without chaos.



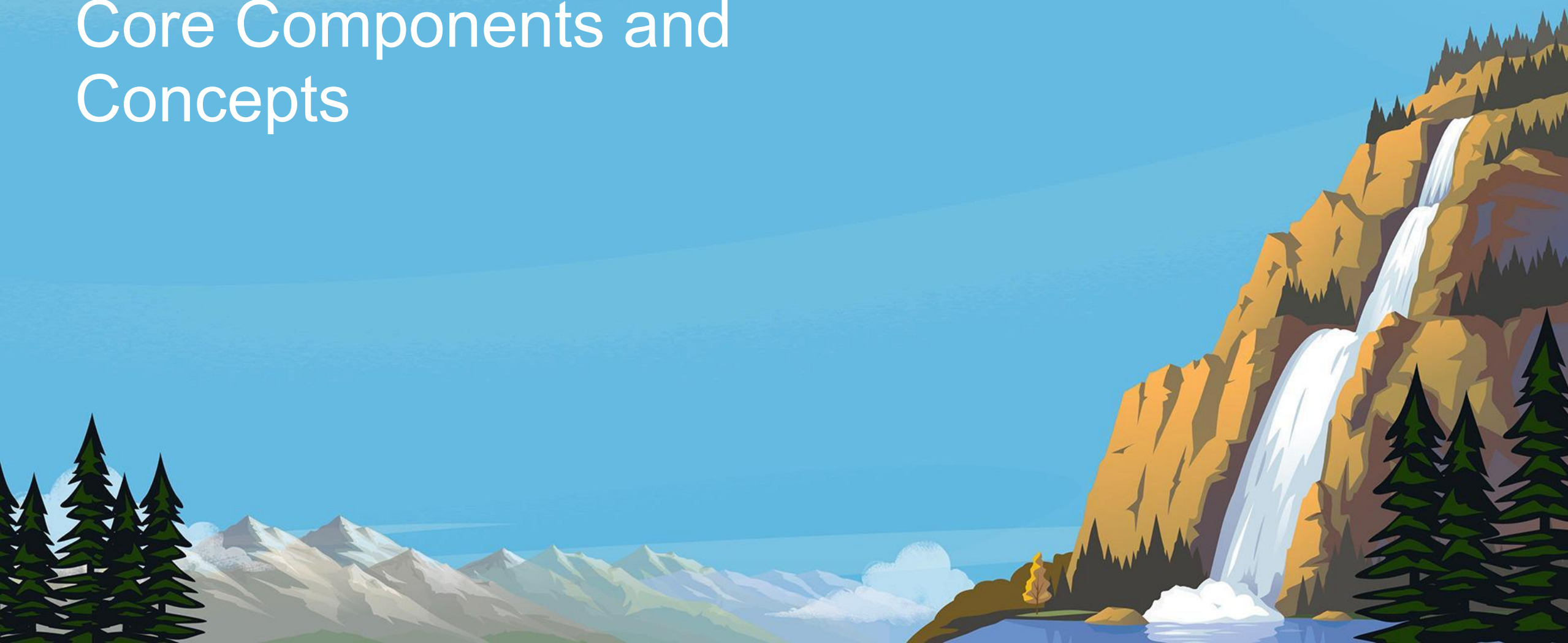

A feature branch is failing, but master is green.

# Virtuous Feedback Cycle

Utility of CI goes up with good development practices.

Value of good practices increases and becomes visible with CI.

# Core Components and Concepts

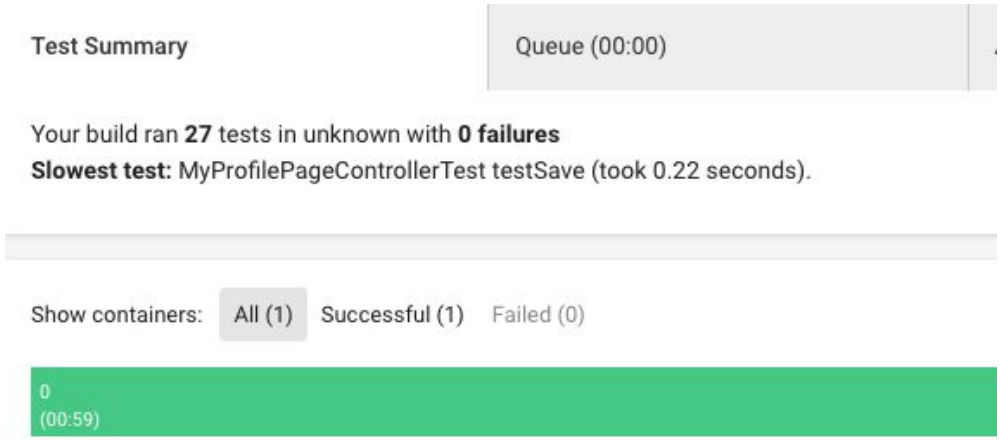


# CI Flow: User Perspective

Write Code → Push → Get Feedback

```
$ git push
```

This is where we're digging in

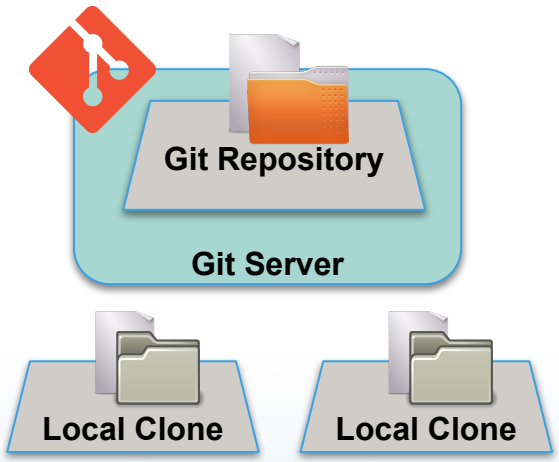


# Core Components of CI with Salesforce DX

## Version control

Server houses master repo, triggers CI process

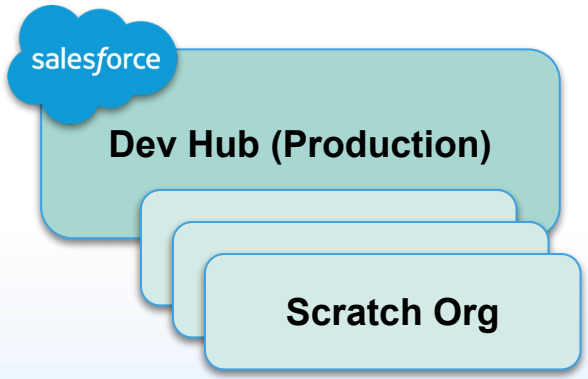
Developer repos hold working copy, connect to scratch orgs




## Salesforce DX

Production is the Dev Hub (controls scratch orgs)

Scratch orgs (~ Salesforce containers) for development and testing

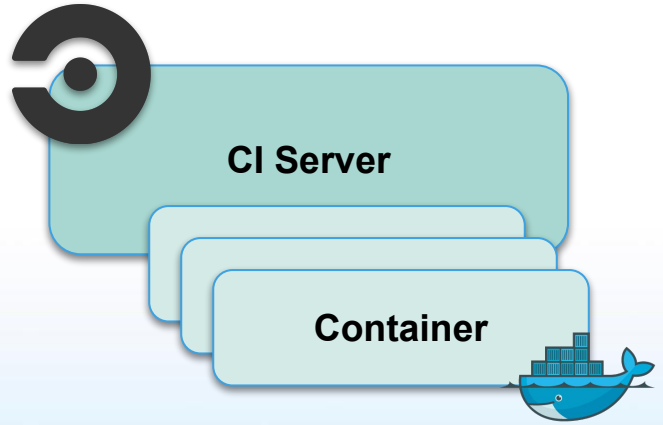



## CI provider

Watches Git server; runs flow on push

Executes steps defined by script

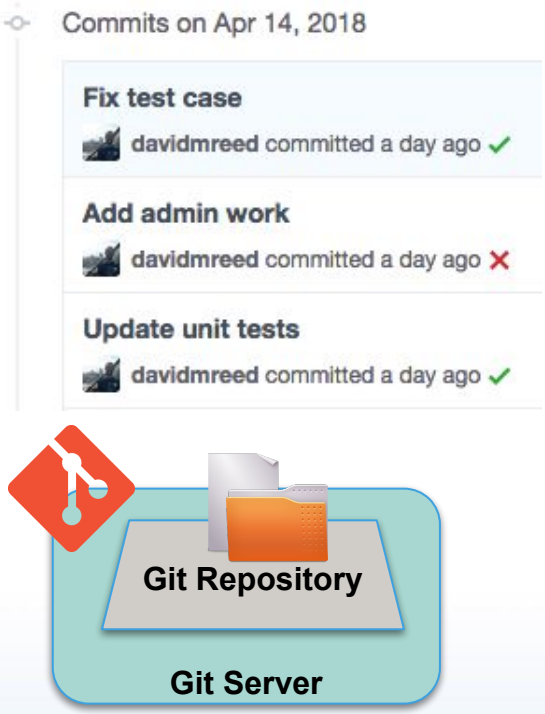
Build runs in virtualized environment, uses scratch org



# Version Control: Source of Truth

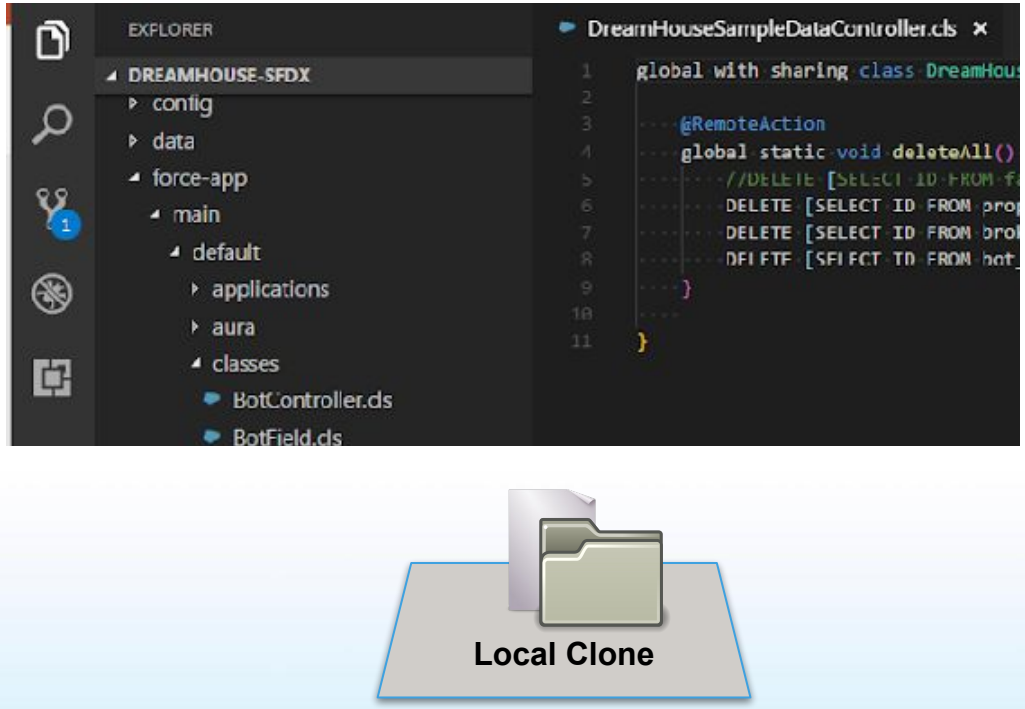
`master` lives on your version control server.

This is the team's source of truth.

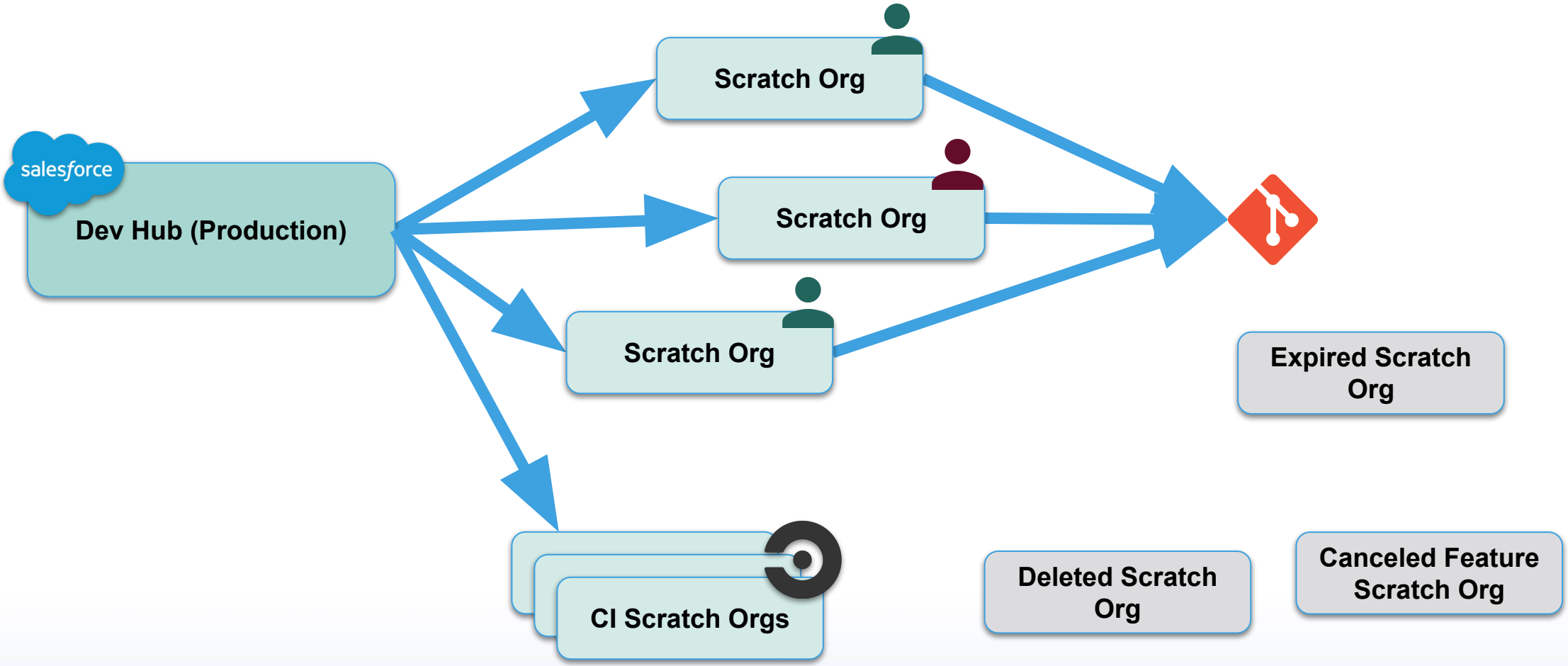



Your current work is built in an SFDX scratch org and tracked in your local Git repo.

These changes don't start CI until they're committed to Git and pushed to the server.



# SFDX Scratch Orgs: Location of Work and Testing



# Continuous Integration Provider: The Conductor

CI server reacts to changes in Git

Scope is configurable – run only on master, specific branches, etc.

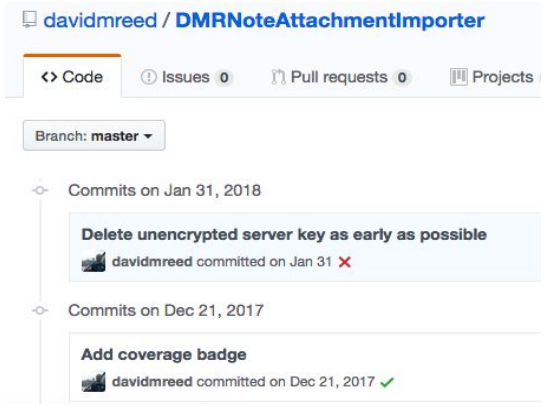
SFDX scratch orgs take over some work done inside a traditional CI solution

CI orchestrates the build/test process

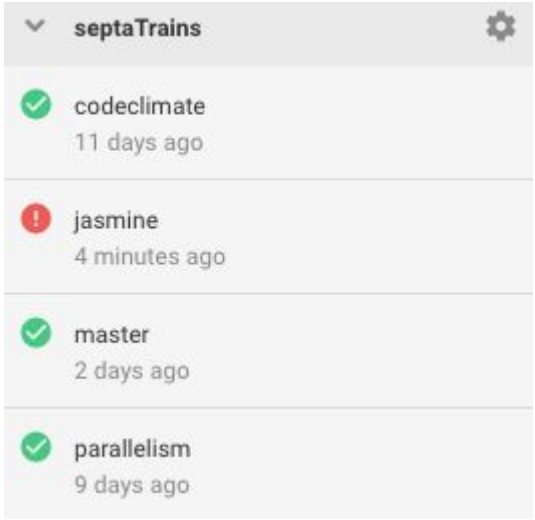
# Results & Visibility

## Current status pushed everywhere you work


Build-level results in CI




Branch-level status in CI

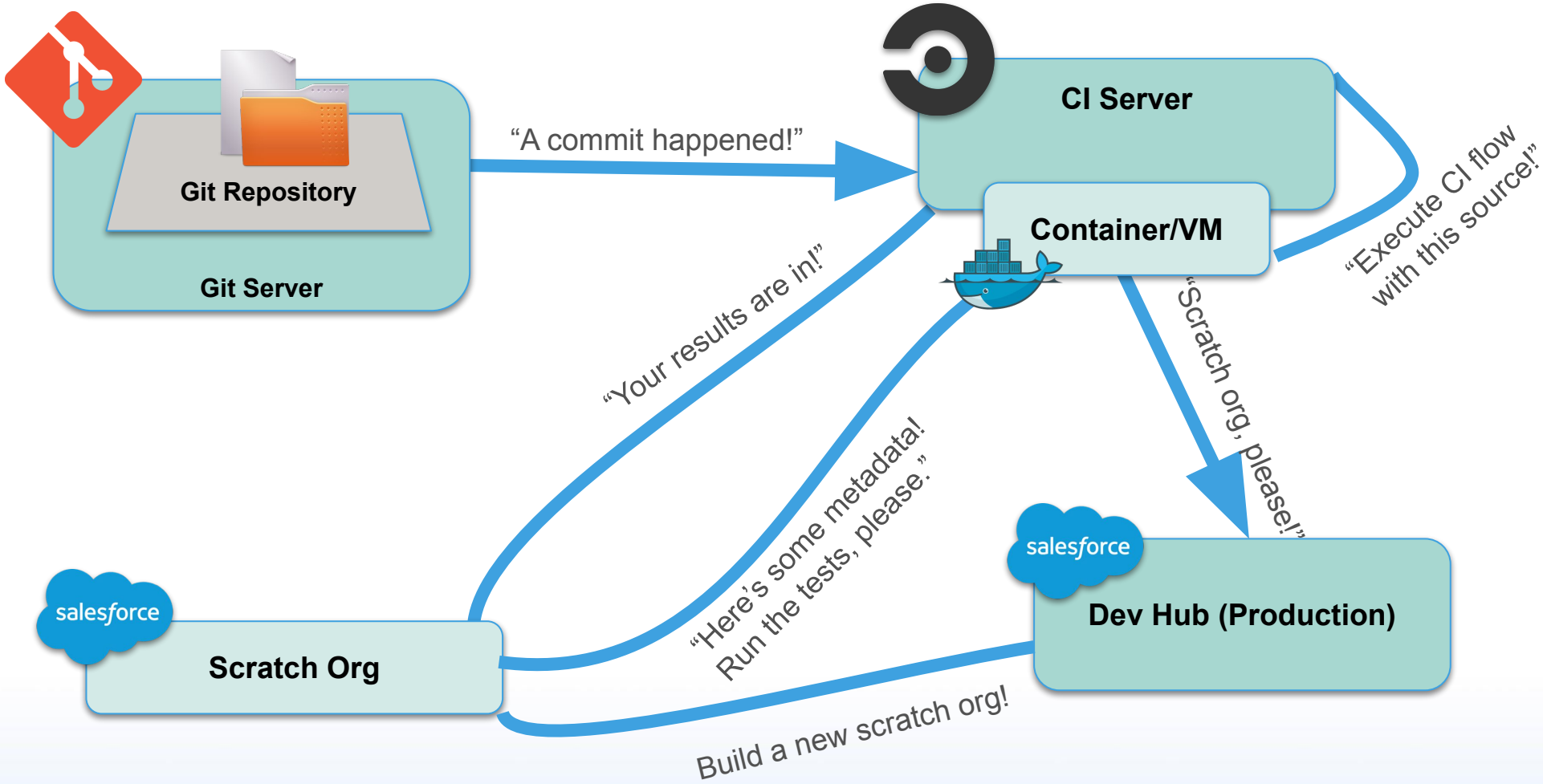



Commit-level status in VCS

# Digging Deeper

## A Systems Perspective

# CI Flow: Connections Between Systems



# Containers, Scripts, and Connections

SFDX and CI rely on containers in different ways

CircleCI runs Bash scripts in Docker virtual environment

SFDX scratch orgs can be thought of like containers for Salesforce

Containerization and scripting allows plugging of toolchain


```
%YAML 1.2
---
YAML: YAML Ain't Markup Language
```


# What Defines an Org?

Broad categories, some overlap

**Edition and Feature Set**

Developer / Enterprise / etc

Optional features like Person Accounts

Things you license from Salesforce – plus some aspects of Setup configuration

**Configuration (Setup)**

Changes you make in Setup that alter the behavior of your org

Feature-level configuration

**Split between JSON scratch org configuration, Metadata API source, and leftovers**

**Source and Metadata**

Apex code

Visualforce

Lightning

Process/Flow/Workflow

Custom objects

**Defined in JSON scratch org configuration**

**Defined in source code (Apex, JS, XML …)**

# Shaping the SFDX Scratch Org

Start with JSON config file.

Deploy source to customize.

Edge-case tweaks with Metadata API or even Selenium.

```
{
    "orgName": "University of Caprica",
    "edition": "Enterprise",
    "features": ["Sites", "Communities", "PersonAccounts"],
    "description": "New architecture for defense mainframe",
    "hasSampleData": "false",
    "orgPreferences" : {
        "enabled": ["NetworksEnabled", "ChatterEnabled"]
    }
}
```

config/project-scratch-def.json


```
- run:
    name: Perform Metadata Deploy
    command: |
        sfdx force:mdapi:deploy -d md-src -w 5
```

Metadata API Deployment

# What Defines a Container?

Broad categories, more overlap

**Operating System**

Typically Ubuntu or another Linux distribution

**Defined by CI setup and/or image selection**

**Software**

Available packages, like SFDX, Java, node.js

**Defined by image selection, scripted in build steps**

**Source and Metadata**

Apex code

Visualforce

Lightning

Process/Flow/Workflow

Custom objects

**Checked out from Git**

# Shaping the CI Container

Pick base image (Java, NPM, etc.)

Customize with `apt-get`, `npm install`, other scripts

Reproducible setup - reproducible results

```
docker:
  - image: circleci/python:3.6.4-node
```

Selecting Docker image in `config.yml`

```
- run:
    name: Install Salesforce DX and simple_salesforce
    command: |
        if [ ! -d node_modules/sfdx-cli ]; then
          export SFDX_AUTOUPDATE_DISABLE=true
          export SFDX_USE_GENERIC_UNIX_KEYCHAIN=true
          export SFDX_DOMAIN_RETRY=300
          npm install sfdx-cli
```

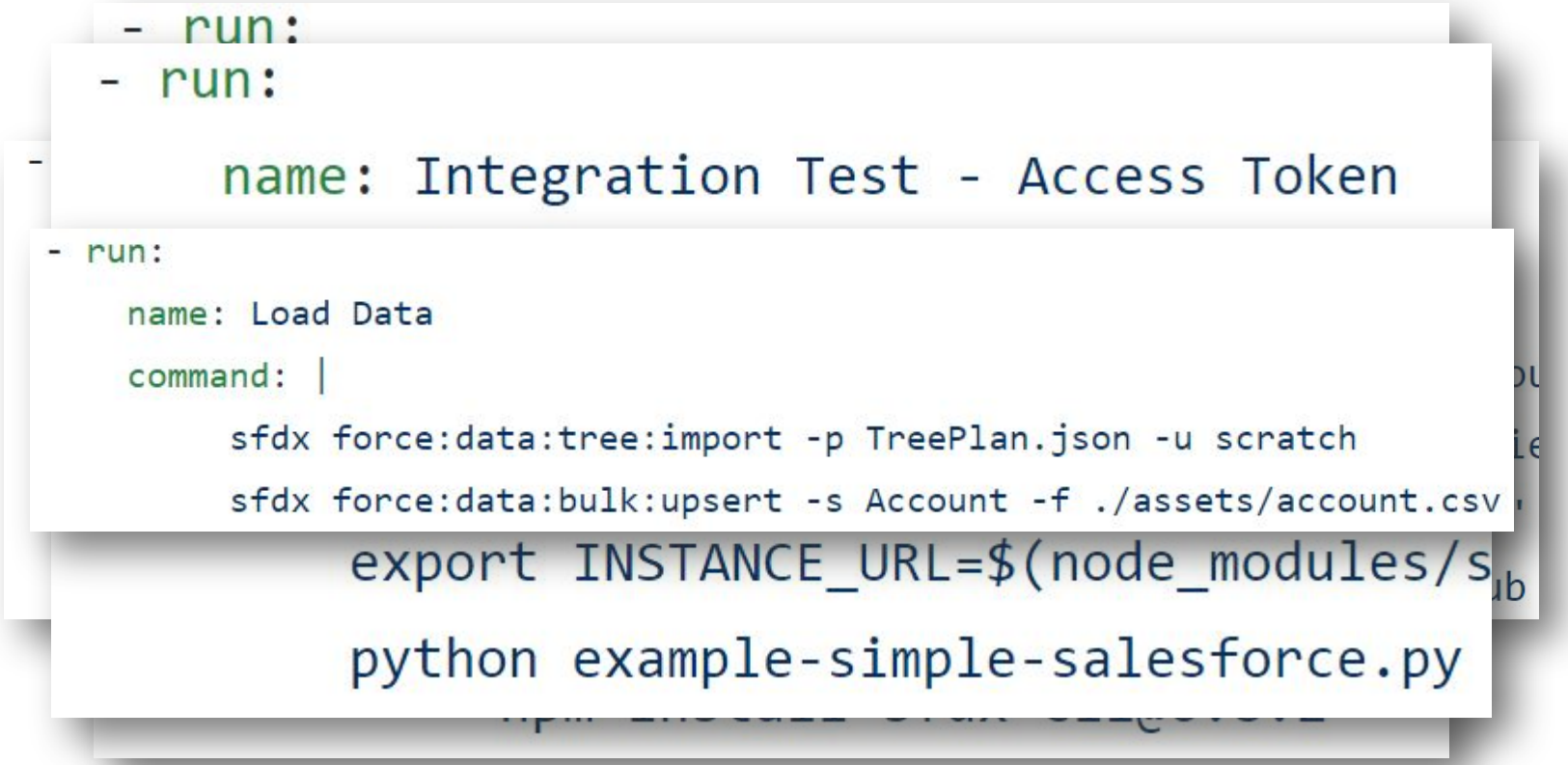
Using `npm` to install `sfdx`

# Scripting Build Steps

(Almost) any tool, any command, any connection

Build steps can:

- Check out source code
- Install software in container
- Connect to SFDX scratch org
- Load data
- Run unit tests
- Run off-platform code
- More to come later



# Key Principles

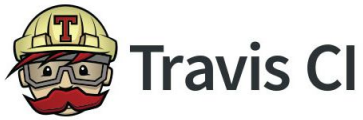
## A bespoke CI process, from off-the-shelf tools

Scripting + containers = **automation** and **reproducibility.**

Once you're scripting with CI, you have enormous power to add tools.

Taking advantage of prior art, then innovating.







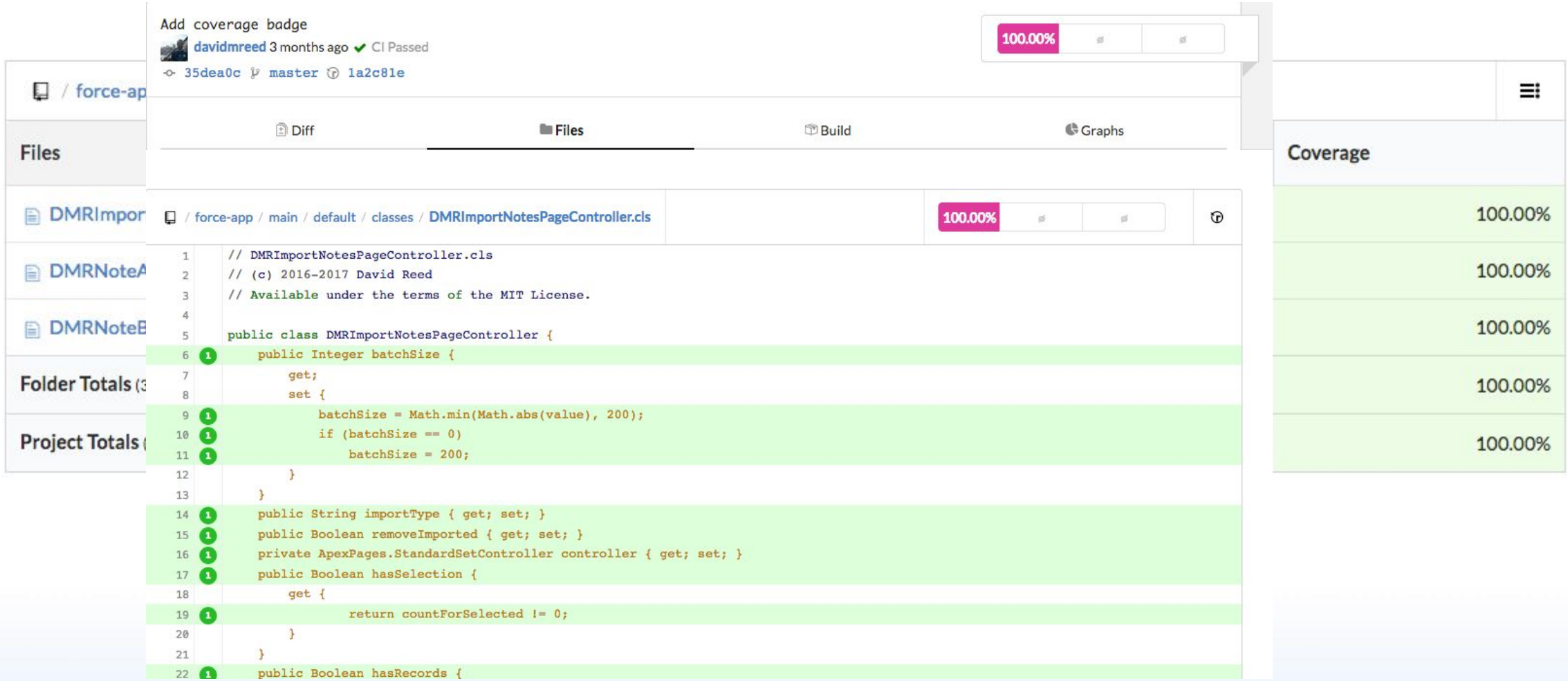


# New Tools

## A Tasting Menu

# Code Coverage Monitoring and Visualization

## Codecov.io

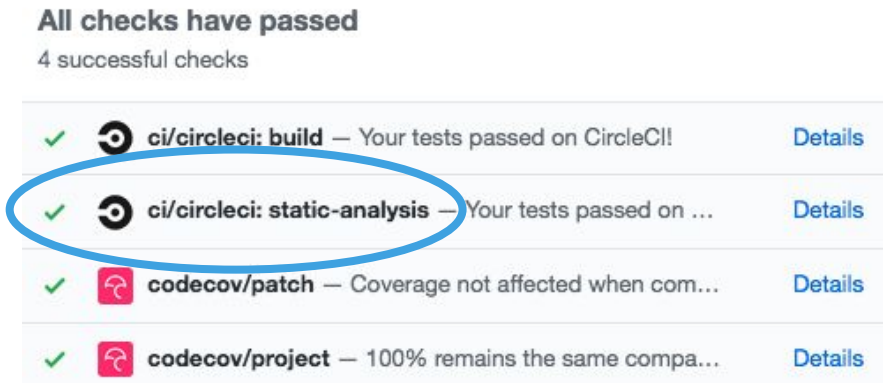


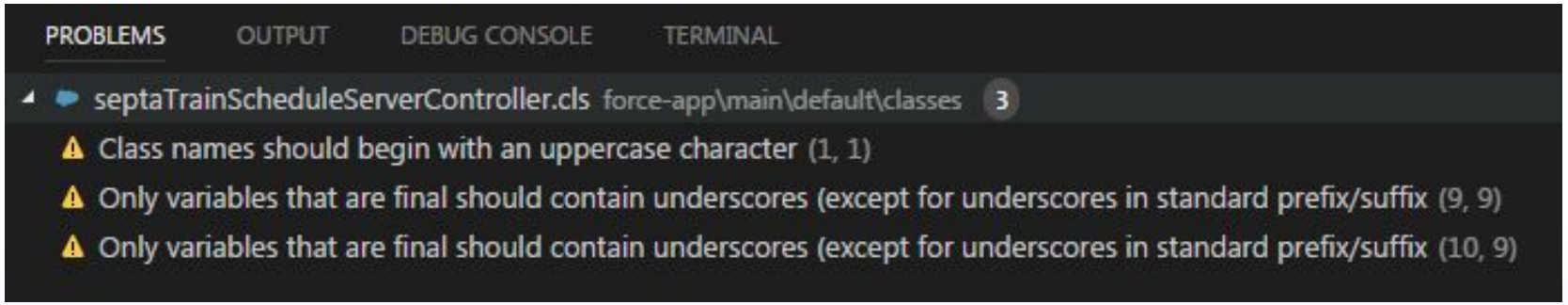
# Static Analysis with PMD

PMD static analysis can run in parallel with scratch org deploy and test.

Block the build on SOQL/DML in loops, more.

PMD can run in your IDE or in the cloud.



# Lightning Testing Service

## Unit test JavaScript with Mocha or Jasmine in SFDX scratch orgs

Jasmine 2.6.1

• • • • • •

```
6 specs, 0 failures

septaTrainTrackerController
  sets selected entity upon event receipt
  updates the map view upon navigate

septaStationScheduleController
  sets schedule and display name upon selection

septaScheduleSidebarController
  sets selected entity upon event receipt

septaEntitySelectorController
  fires event upon navigate
  sets select options from the controller
```


# Testing Multiple Org Types and Shapes

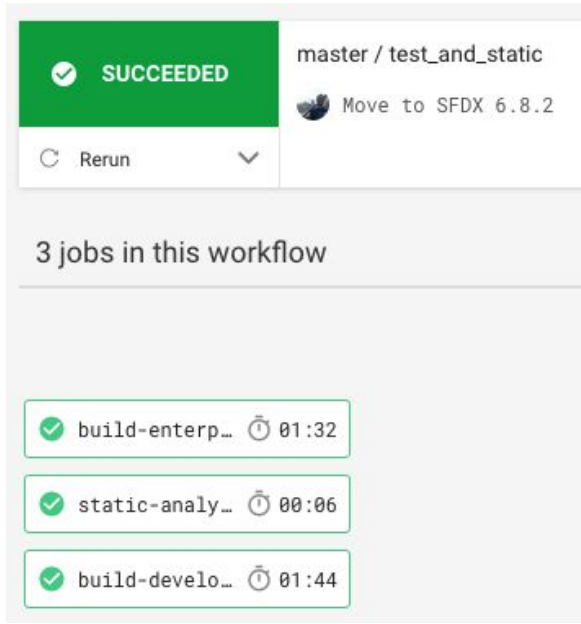
project-scratch-def.json

```
{
    "orgName": "David Reed",
    "edition": "Enterprise",
    "features": ["Sites"],
    "orgPreferences" : {
        "enabled": ["S1DesktopEnabled"]
    }
}
```

Scratch definition JSON file(s)

```
workflows:
  version: 2
  test_and_static:
    jobs:
      - build-enterprise
      - build-developer
      - static-analysis
```

Workflow definition




Results per org type in CircleCI and GitHub

# Testing Software that Integrates with Salesforce


Salesforce DX isn't limited to Apex code running on the Force.com platform.

Code that integrates to Salesforce can be tested against scratch orgs too.

```
- run:
    name: Integration Test - Username and Password
    command: |
        . venv/bin/activate
        node_modules/sfdx-cli/bin/run force:user:password:generate > /dev/null
        export PASSWORD=$(node_modules/sfdx-cli/bin/run force:user:display --json | pyth
        export SF_USERNAME=$(node_modules/sfdx-cli/bin/run force:user:display --json | p
        python example-simple-salesforce.py -p "$PASSWORD" -u "$SF_USERNAME" -t "" -s
```

Integration testing a Python/simple_salesforce application

# Building Packages for Continuous Deployment

“Gate” build steps conditioned on successful tests or approval

Build in sophisticated branching and tagging strategy

Create new Unlocked Package versions for deployment

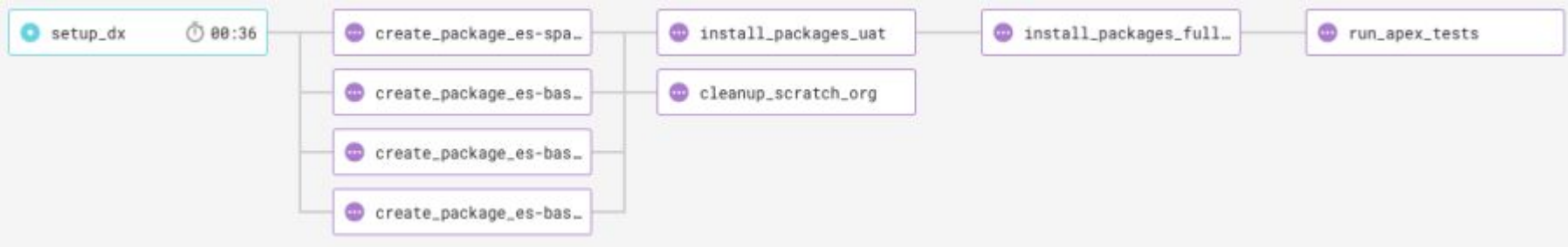



Image via Salesforce Developer Blog (https://developer.salesforce.com/blogs/2018/06/working-with-modular-development-and-unlocked-packages-part-4.html)

# Links and Resources

**Getting Started with Salesforce DX** (5-part series):
`developer.salesforce.com/blogs/2018/02/getting-started-salesforce-dx-part-1-5.html`

**Continuous Integration with Salesforce DX Trailhead**:
`trailhead.salesforce.com/en/modules/sfdx_travis_ci`

**Templates on GitHub**: `github.com/davidmreed/circleci-sfdx-examples`
(*includes all code from this presentation)*

**David Reed**:

www.ktema.org david@ktema.org @aoristdual


THANK YOU