PIPELINES && SERVERLESS && AUTOMATION, OH MY!

AN INTRO TO THE DEV SIDE OF SECURITY

Cassandra Young
Jonn Callahan
Evan Perotti

APIS, SERVERLESS & MICROSERVICES

CASSANDRA YOUNG (AKA MUTEKI)

- Senior Scientist (R&I - Cloud Security) @ Security Risk Advisors
- Grad Student in Computer Science @ University of Pennsylvania
- Blue Team Village Organizer
- Lives for international travel, scuba diving, woodworking, jigsaw puzzles and baking

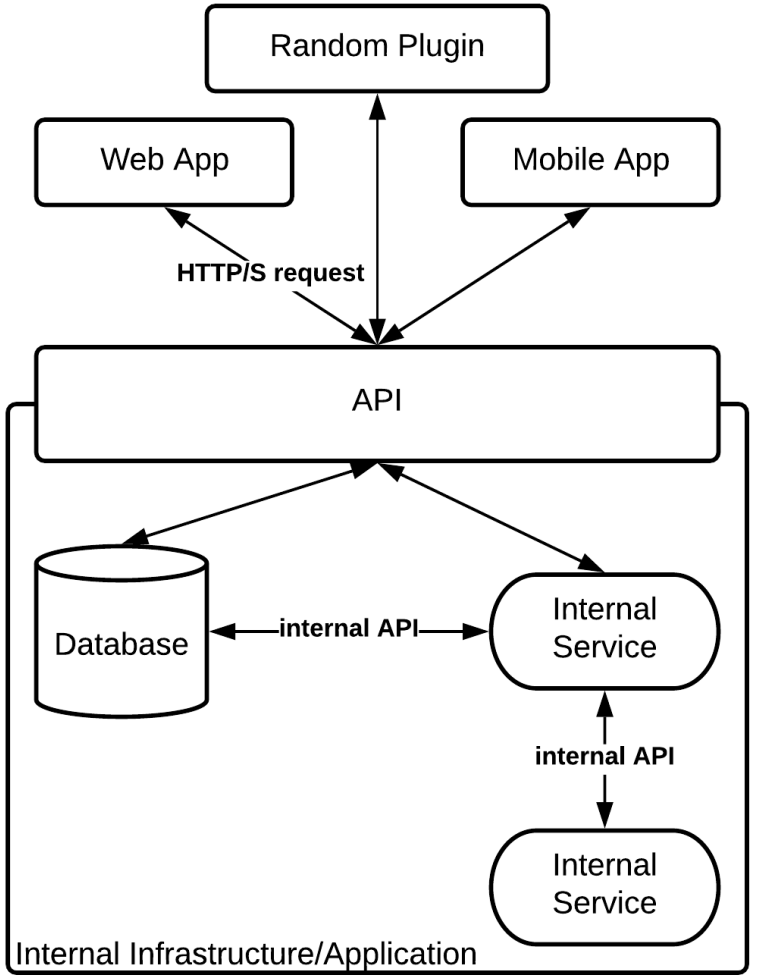
Cassandra Young (she/her)

Twitter: @muteki_rtw

GitHub: github.com/muteki-apps

API DEFINITION

- TL;DR: provides standardized, programmatic method of interaction between backend logic / software and external services
- Layer of abstraction to simplify interaction with a service
  - Common Functionality: GET, PUT, POST, DELETE
- Implementation on the backend varies widely depending on the service



API USE

- Connect to APIs w/ HTTP requests, using any programming language
- Examples: request headers w/ auth, JSON body, Request & Response Headers
- Anatomy of a REST API request:
  - Endpoint – URL specifying the data you're interacting with
  - Method – how you're interacting with it
  - Headers – metadata include with request & response, can include auth
  - Data/Payload – if creating/updating, what info to add/change (usually JSON formatted)

```
41
42  endpoint = 'https://api.somenerdyapp.com/stuff/earlgreytea'
43  request_headers = {'user-agent': 'borg_app/0.0.1'}
44
45  response = requests.get(endpoint, headers=request_headers)
46  response.headers
47  '''
48  {
49      'content-encoding': 'gzip',
50      'transfer-encoding': 'chunked',
51      'connection': 'close',
52      'server': 'svr/1.0.4',
53      'x-runtime': '148ms',
54      'etag': '"e1ca502697e5c9317743dc078f67693f"',
55      'content-type': 'application/json'
56  }
57  '''
58
```

API SECURITY CONSIDERATIONS

Risk

- String injection
- Unauthorized data access (ex. IDOR)
- D/DoS, Man in the Middle attacks, complex attacks

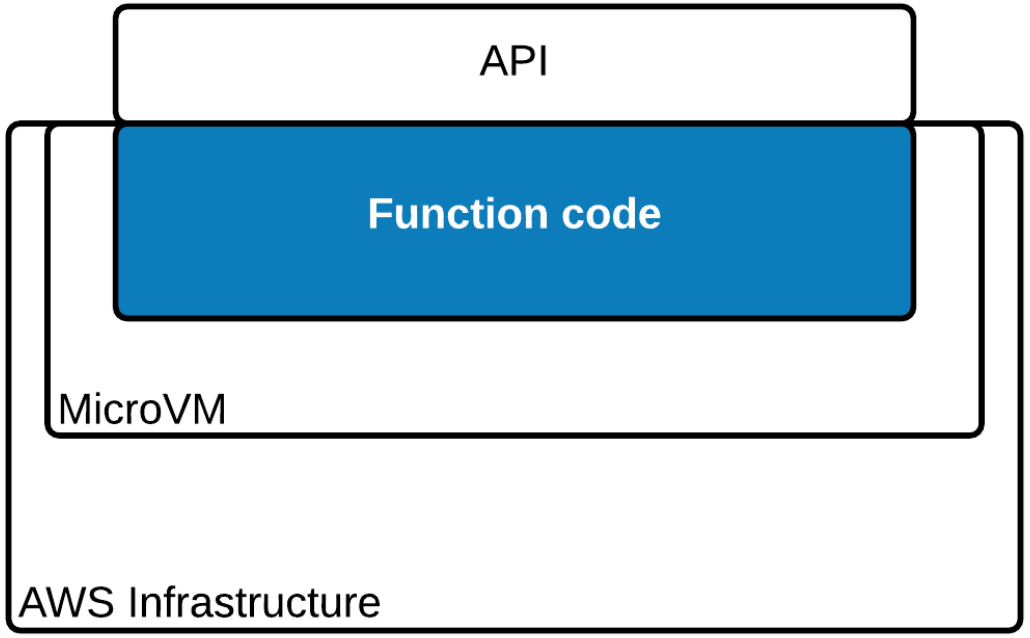
Mitigation

- Input validation
- Authentication/authorization, data access controls
- Web Application Firewall, TLS, rate limiting

Resources:
ShellCon talk @ 14:30 on API Security!
API Hacking talk: https://www.youtube.com/watch?v=qqmyAxfGV9c
Use Postman to poke at APIs: https://www.postman.com/

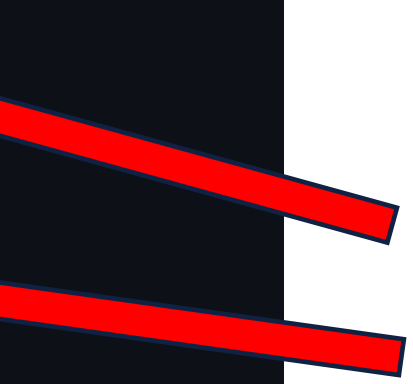
SERVERLESS

- Execution model abstracting away infrastructure
- Code-focused, obfuscating OS & dependencies
- Simplified development process
- Event-driven, primarily accessed via APIs
- Stateless, no persistent storage



SERVERLESS FUNCTION EXAMPLE

```
3   import os
4   import json
5   import boto3
6   import botocore
7
8   def lambda_handler(event, context):
9       # accepted actions:
10          # request_keys
11          # admin: initialize
12          # admin: rotate_all
13
14      result = "no results"
15      success = False
16
17      if "request_keys" in event.keys():
18          result = get_node_keys()  # returns dict of nodes and
19          if result:
20              print(result)
21              success = True
22      elif "admin" in event.keys():
23          if event["admin"].lower() == "initialize_all":
```



- Supported **library** installs done for the developer
- JSON-formatted **event** is the input to the function
- Context object provides info about the runtime env and other metadata
- Program logic up to developer
- Function will return JSON-formatted response to caller

SERVERLESS FUNCTION EXAMPLE

- Invoking one AWS Lambda from another via internal API
- AWS boto3 library helps interface with the API
- Other Lambda must have policy that allows it to be invoked

```
def send_message(msg, dest_lambda):
    client = boto3.client('lambda')
    response = client.invoke(
            FunctionName=dest_lambda,
            InvocationType='RequestResponse',
            LogType='Tail',
            Payload=json.dumps(msg),
        )
    try:
        print(json.load(response['Payload']))
```

SERVERLESS FUNCTION EXAMPLE: IAM PERMISSIONS ON LAMBDA

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBList",
            "Effect": "Allow",
            "Action": [
                "dynamodb:List*",
                "dynamodb:DescribeTimeToLive",
                "dynamodb:DescribeReservedCapacity*",
                "dynamodb:DescribeLimits"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:[redacted]:table/ta_node_1_db",
                "arn:aws:dynamodb:us-east-1:[redacted]:table/ta_directory_db"
            ]
        },
        {
            "Sid": "DynamoDBModify",
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateItem",
```

- Example of scoped policy
- Restricting AWS Lambda permissions reduces attack surface
- IAM/permissions is the serverless security perimeter!

```
comp = input('\nYour computation? => ')
if not comp:
   print ("No input")
else:
   print ("Result =", eval(comp))
```

The (Python 3) `input` function reads a line from standard input. The `eval` function performs the evaluation of the expression provided. Given that `eval` evaluates the input as a Python expression, it can also calculate values if you prefer. For example, if the input is `30 * 12 + 5` then it computes the value and outputs: `Result = 365`.

However, Python expressions can do a lot of things, for example, consider this:

```
__import__('os').system('rm -rf /')
```

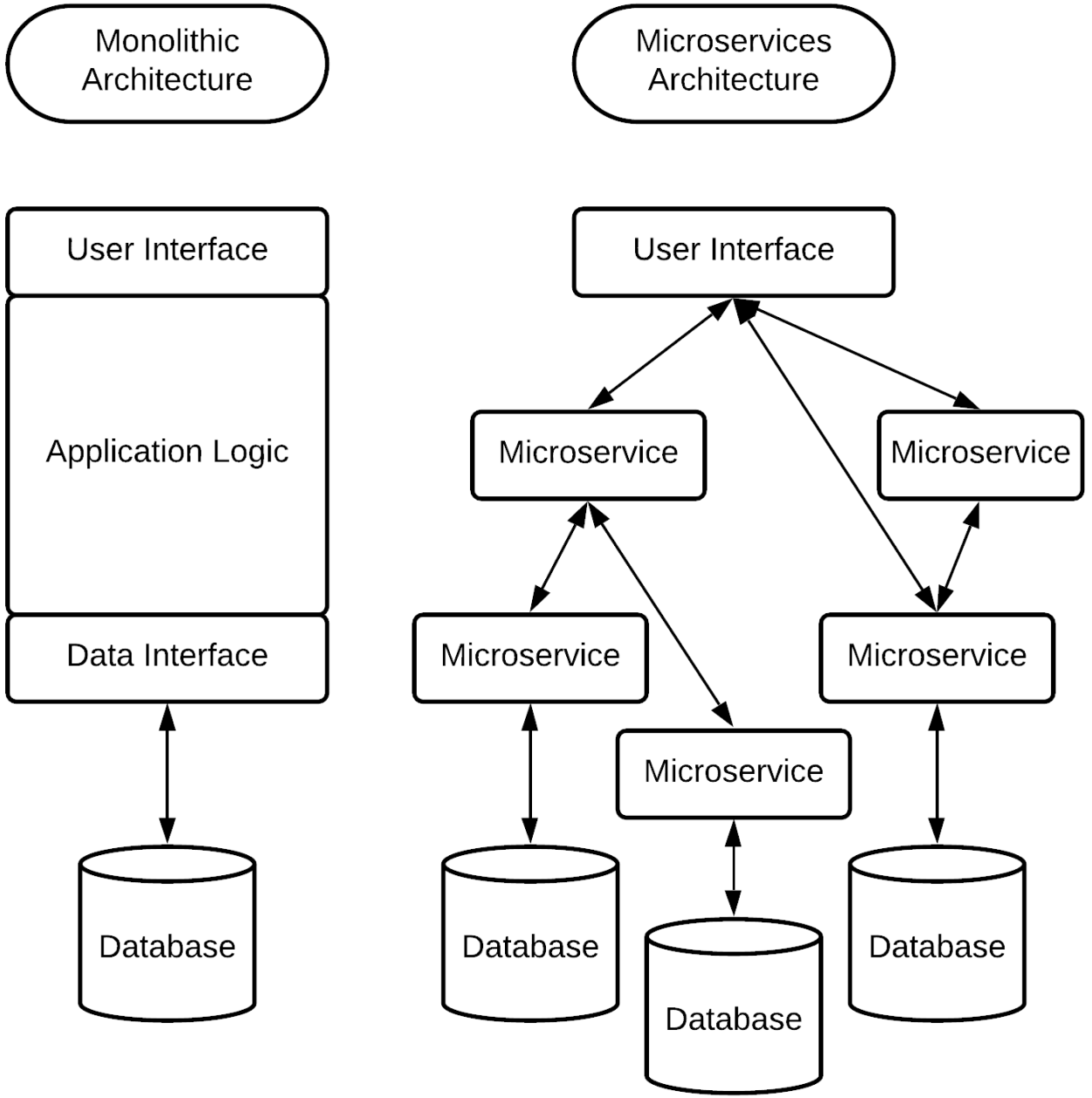
SERVERLESS FUNCTION COMPROMISE EXAMPLE

- String injection
- Access to Environment
- Enumerate Lambda's Roles/Policies
- Privilege Escalation / Resource Access
- IAM / Permissions = Security Perimeter

Image credit: https://research.cs.wisc.edu/mist/SoftwareSecurityCourse/Chapters/3_8_3-Code-Injections.pdf

MICROSERVICES

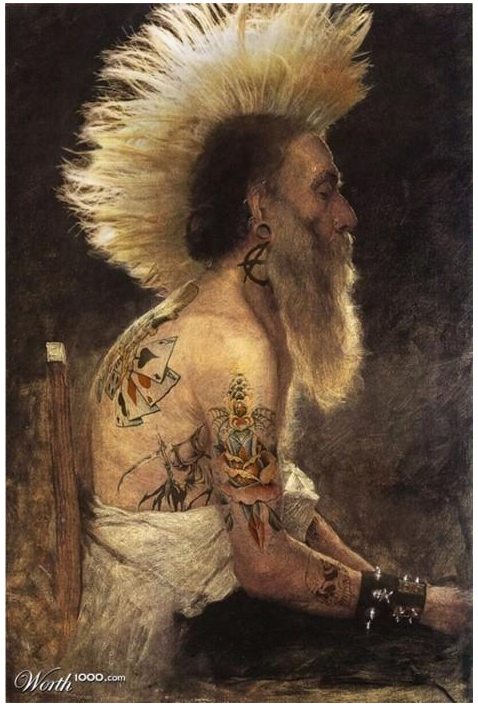
- Microservices Architecture is a design pattern
- Collection of loosely-coupled components
  - Commonly implemented using containers or serverless functions
  - APIs connect components
- Development and upgrade focus shifted to component level



PIPELINES

JONN CALLAHAN (ATTICUSS)

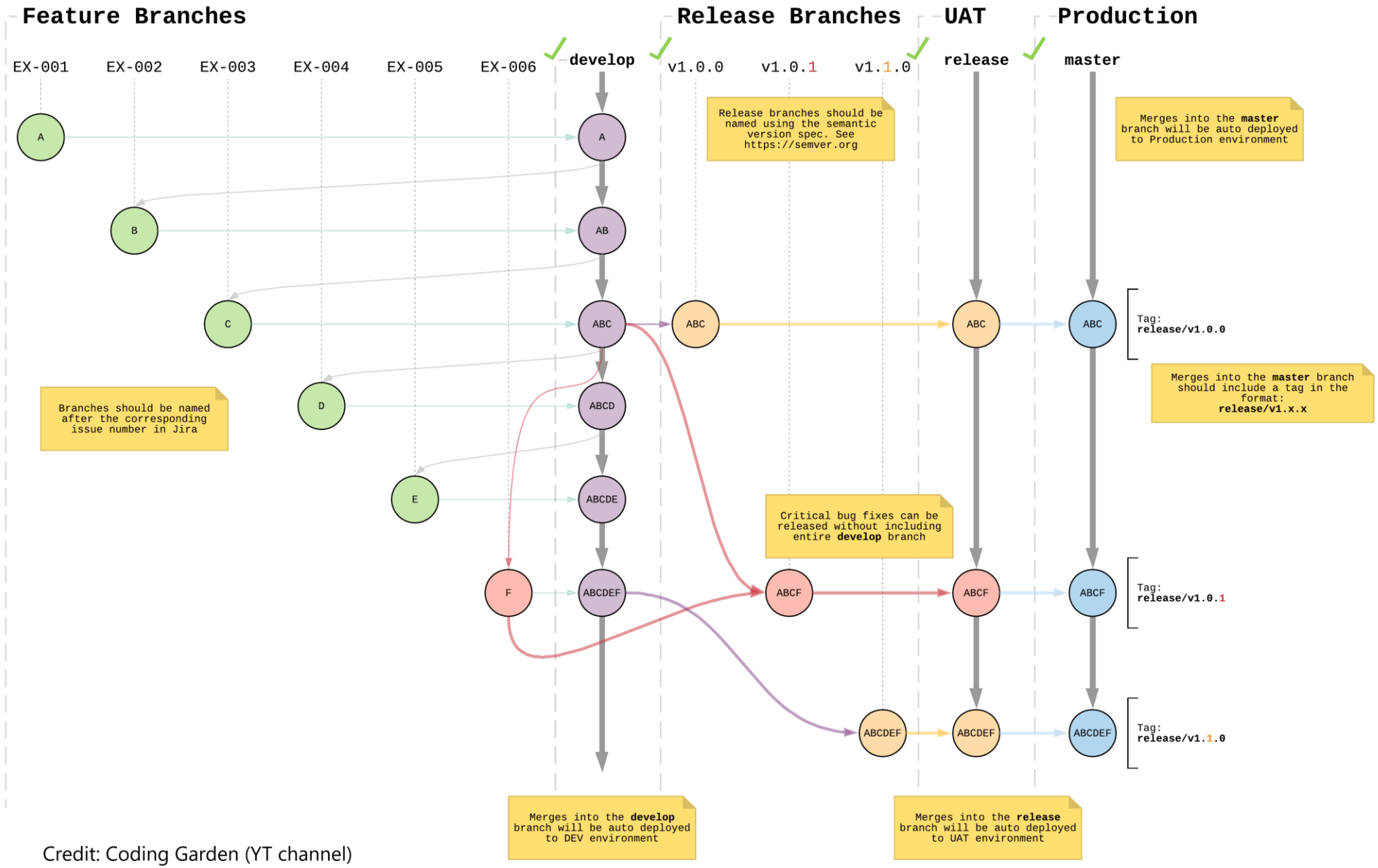
- Lead Scientist / Security Risk Advisors
- Python is great, as is Golang
- Distributed computing is fascinating
- Metalhead \m/




WHY DON'T WE
TAKE OUR CODE
AND DEPLOY
IT OVER THERE
imgflip.com

Branching strategy:
develop - includes latest features / fixes
v1.x.x - created in preparation of release to UAT environment
release - the current state of the UAT environment
master - the curent state of the production environment

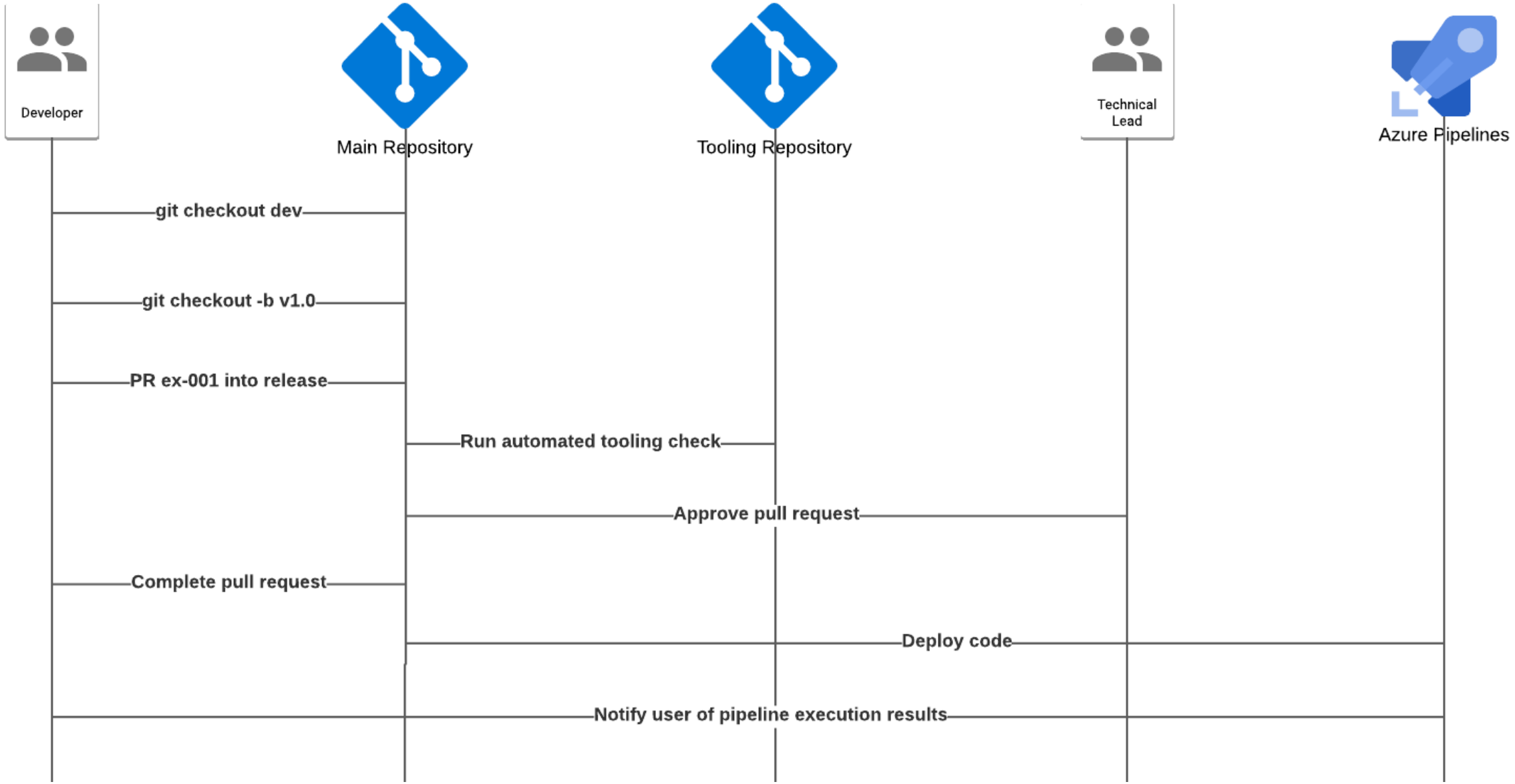
Merge requests into the **develop**, **release** and **master** branches must pass tests, code review and QA before being merged.



Credit: Coding Garden (YT channel)

TYPICAL FLOW CONTROLS

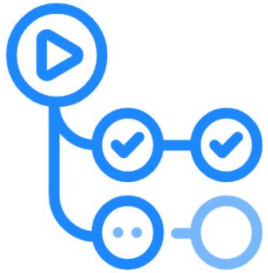
- Implemented against pull requests
  - Manual approvement gateways
  - Automated tests
- Branch protections
  - Limits or entirely blocks ability to push directly to sensitive branches
  - Limit the files that can be modified by a pull requests
- Pipeline protections
  - Limit who can modify the pipeline configuration
  - Limit who can modify automated test code


Developer
Main Repository
Tooling Repository
Technical Lead
Azure Pipelines
git checkout dev
git checkout -b v1.0
PR ex-001 into release
Run automated tooling check
Approve pull request
Complete pull request
Deploy code
Notify user of pipeline execution results

CONCEPTUALLY

- Covers the operational gap between development and deployment
- Uses automation to reduce friction
- Takes freshly written code and moves it through higher environments until its released in production
  - git branching and pull requests

Buildkite

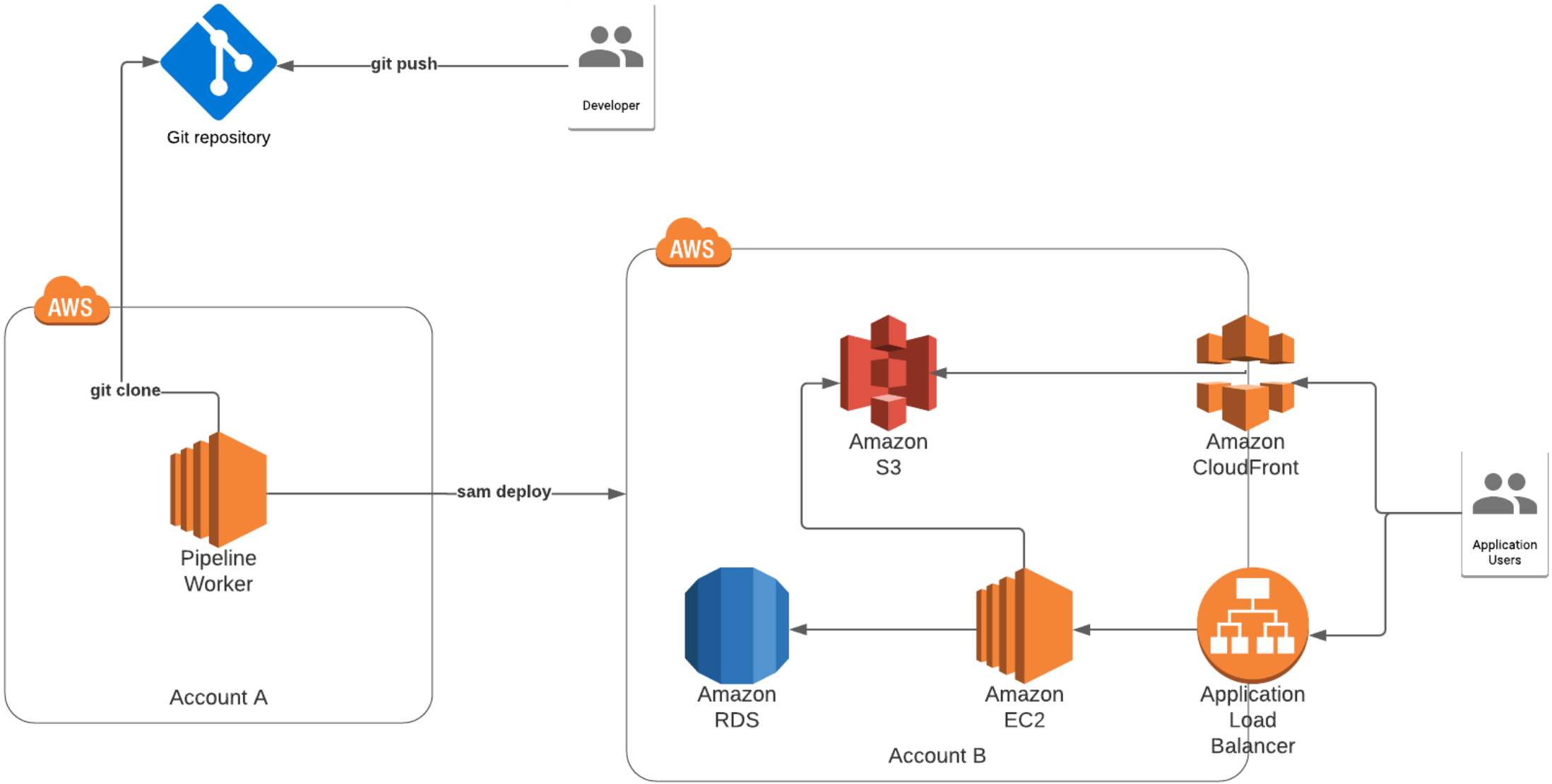


GitHub Actions

concourse

Jenkins

Azure DevOps

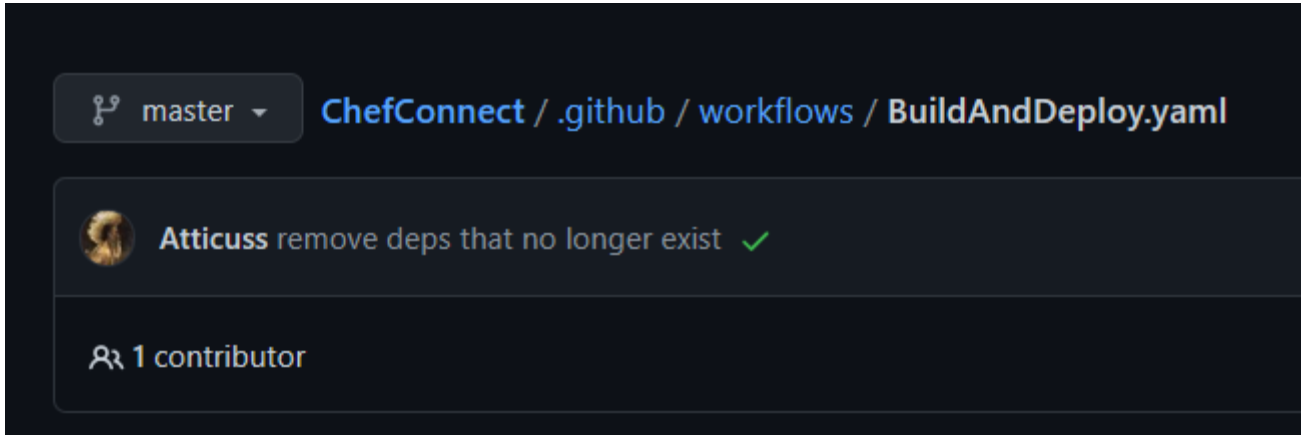

Git repository
git push
Developer
AWS
git clone
Pipeline
Worker
Account A
sam deploy
AWS
Amazon
S3
Amazon
CloudFront
Application
Users
Amazon
RDS
Amazon
EC2
Application
Load
Balancer
Account B

CONCRETELY (MODERN VARIANTS)

- Utilize configuration files for controlling behavior (usually YAML)
- Trigger on various modifications to a git repo, such as a push or new pull request
- Pipeline triggers contain n jobs for doing Things™, e.g.:
  - Deploy to production environment
- Each job runs inside a container (sequential or parallel)
- Each job contains n discrete tasks/steps/etc, e.g.:
  - sam deploy
- Containers are pretty ubiquitous at this point
  - Allows for isolation and flexibility

FOR THOSE PLAYING ALONG AT HOME

- https://github.com/atticuss/ChefConnect
  - Yes, the code is horrible – was my first Go app and I've learned a lot since then
  - Github Actions are configured via /.github/workflows/foo.yaml




action trigger via whitespace removal Build and Deploy #75
Summary
Jobs
Deploy to Dev
Deploy to Prod
Deploy to Dev
succeeded 15 seconds ago in 3m 20s
Set up job
Source checkout
Setup Golang env
Setup python
Install dependencies
Configure AWS credentials
Configure private MDBPro SSH key
Build infra/code then deploy
Post Configure private MDBPro SSH key
Post Configure AWS credentials
Post Source checkout
Complete job

75 lines (65 sloc) | 2.66 KB

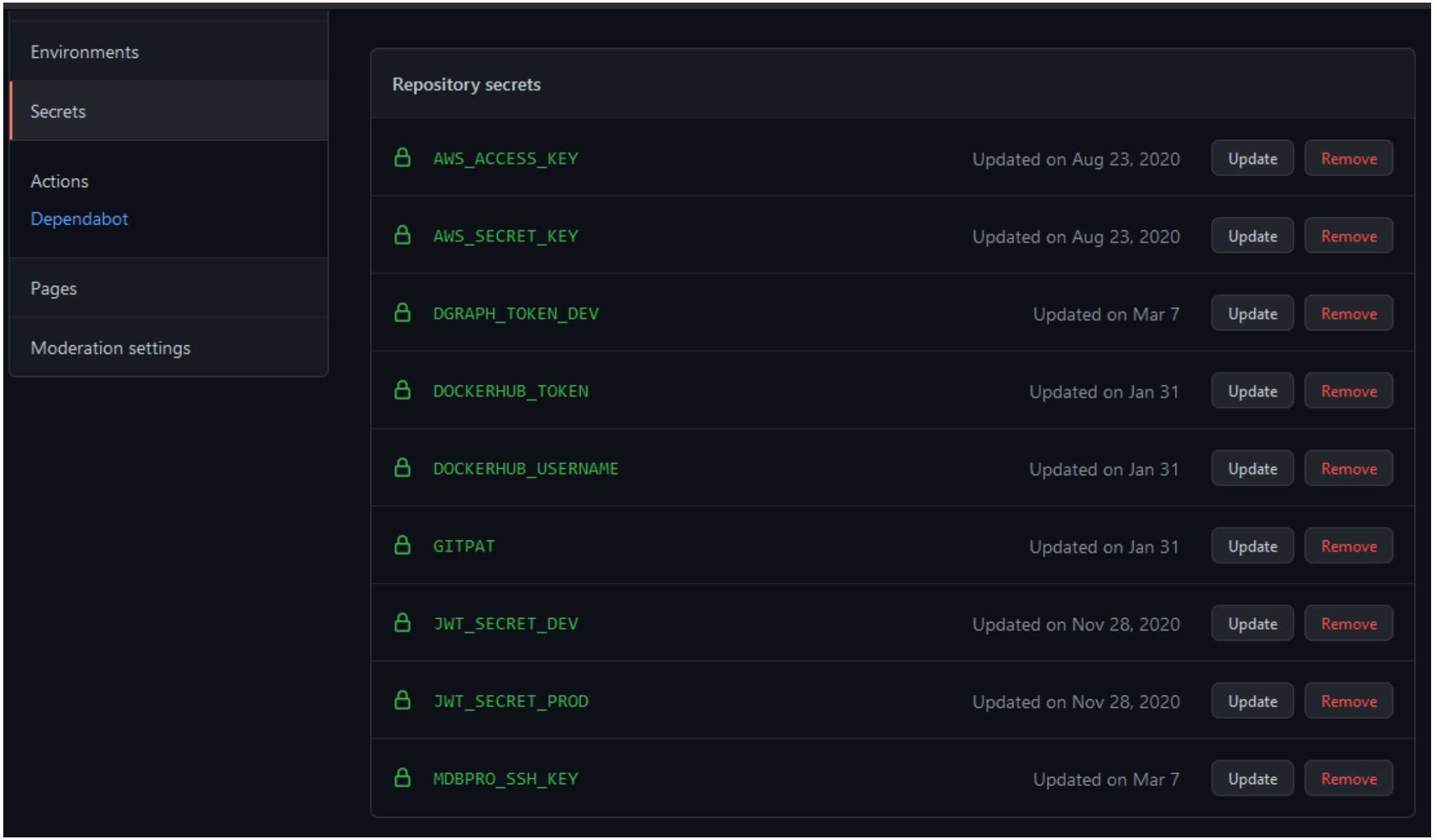
```yaml
name: Build and Deploy
on:
  push:
    branches:
      - dev
      - prod
```

```yaml
jobs:
  dev-deploy:
    name: Deploy to Dev
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/dev'
    steps:
      - name: Source checkout
        uses: actions/checkout@master

      - name: Setup Golang env
        uses: actions/setup-go@v1
        with:
          go-version: "1.14.4"

      - name: Setup Python
        uses: actions/setup-python@v1
        with:
          python-version: 3.8
```

```
26        - name: Install dependencies
27          run: |
28            pip install aws-sam-cli && sudo apt update && sudo apt install build-essential
29
30        - name: Configure AWS credentials
31          uses: aws-actions/configure-aws-credentials@v1
32          with:
33            aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY }}
34            aws-secret-access-key: ${{ secrets.AWS_SECRET_KEY }}
35            aws-region: us-east-1
36
37        - name: Build to Dev
38          run: |
39            make buildLambda
40            sam package --template-file template.yaml --output-template-file packaged.yaml --s3-
41            sam deploy --template-file packaged.yaml --stack-name sam-chefconnect-dev --capabili
```


Environments
Secrets
Actions
Dependabot
Pages
Moderation settings
Repository secrets
AWS_ACCESS_KEY
Updated on Aug 23, 2020
Update
Remove
AWS_SECRET_KEY
Updated on Aug 23, 2020
Update
Remove
DGRAPH_TOKEN_DEV
Updated on Mar 7
Update
Remove
DOCKERHUB_TOKEN
Updated on Jan 31
Update
Remove
DOCKERHUB_USERNAME
Updated on Jan 31
Update
Remove
GITPAT
Updated on Jan 31
Update
Remove
JWT_SECRET_DEV
Updated on Nov 28, 2020
Update
Remove
JWT_SECRET_PROD
Updated on Nov 28, 2020
Update
Remove
MDBPRO_SSH_KEY
Updated on Mar 7
Update
Remove

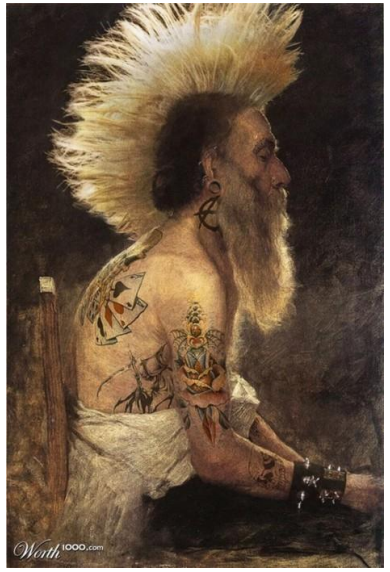

Summary
Jobs
Deploy to Dev
Deploy to Prod
Deploy to Dev
succeeded 15 seconds ago in 3m 20s
Search logs
139 SAM CLI update available (1.33.0); (1.19.1 installed)
140 To download: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html
141 Uploading to 5366adcdb089260f63540c9250010d09.template 5920 / 5920 (100.00%)
142
143 Deploying with following values
144 ==========
145 Stack name : chefconnect-dev
146 Region : us-east-1
147 Confirm changeset : False
148 Deployment s3 bucket : None
149 Capabilities : ["CAPABILITY_AUTO_EXPAND", "CAPABILITY_IAM", "CAPABILITY_NAMED_IAM"]
150 Parameter overrides : {"Environment": "dev", "JwtSecretKey": "***", "DgraphHost": "dev.dgraph.dc1.veraciousdata.io", "DgraphPort": "9080", "DgraphAuthToken": "***"}
151 Signing Profiles : {}
152

(AB)USE CASES

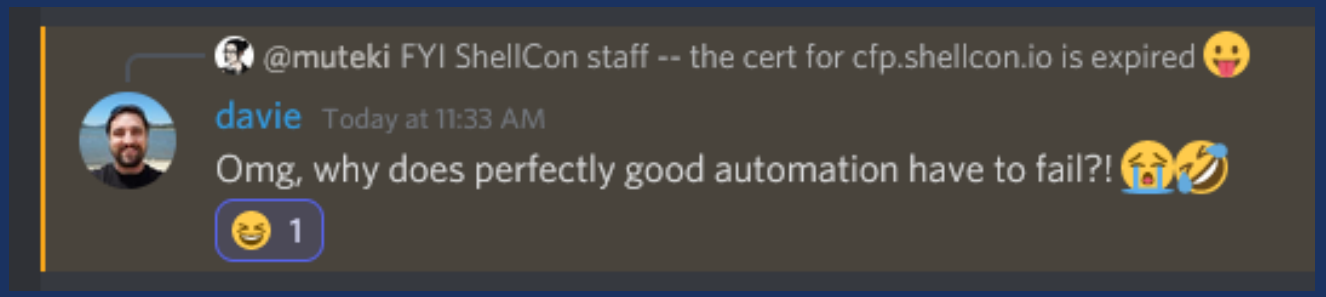
- Introduce malicious code into dev branch
- Compromise the pipeline itself
  - Jenkins boxes are ripe
  - Poison automated check tooling
  - Poison pipeline configuration file
    - Injection attacks are fair game!
  - Steal pipeline credentials (the ones used for performing deployments)
  - Steal pipeline secrets
- Compromise the pipeline infrastructure
  - Gain access to underlying VMs, k8s nodes, etc.
    - Node isolation can help mitigate this
  - Poison container repository (if used)

QUESTIONS?

- Feel free to ping me in Discord in the meantime
  - @atticuss



INFRASTRUCTURE AUTOMATION



EVAN
@2XXEFORMYSHIRT

- Lead Scientist @ Security Risk Advisors (sra.io)
- Focus on offensive security operations

"INFRASTRUCTURE AS CODE" (IAC)

- Store infrastructure and configuration as "code"

- Benefits

  Centralization Version control Standardization

  Auditability Portability*

TERRAFORM

- Popular IAC tool
- Custom language (Hashicorp Configuration Language / "HCL")
- "Data sources" and "Resource"
- Module system
- State management

- Security applications
  - "Shifting left"

Terraform

EXAMPLE: GENERATE RSA KEY + IMPORT

```
# Generating a random key for the project
resource "tls_private_key" "rsakey" {
    algorithm = "RSA"
    rsa_bits = 4096
}

# Import key into AWS
resource "aws_key_pair" "awskey" {
    key_name = "${var.project}-key"
    public_key = tls_private_key.rsakey.public_key_openssh
}
```

EXAMPLE: GENERATE RSA KEY + IMPORT

```
# Generating a random key for the project
resource "tls_private_key" "rsakey" {
    algorithm = "RSA"
    rsa_bits = 4096
}

# Import key into AWS
resource "aws_key_pair" "awskey" {
    key_name = "${var.project}-key"
    public_key = tls_private_key.rsakey.public_key_openssh
}
```

EXAMPLE: GENERATE RSA KEY + IMPORT

main.tf

```
# Generating a random key for the project
resource "tls_private_key" "rsakey" {
    algorithm = "RSA"
    rsa_bits = 4096
}

# Import key into AWS
resource "aws_key_pair" "awskey" {
    key_name = "${var.project}-key"
    public_key = tls_private_key.rsakey.public_key
}
```

terraform.tfvars

```
project = demo
```

value

definition

vars.tf

```
variable "project" {
    type = string
    description = "Project prefix for assets"
    validation {
      condition = can(regex("^[a-zA-Z0-9\\-]+$", var.project))
      error_message = "Project name can only consist of alphanumerics and dashes."
    }
}
```

EXAMPLE: GENERATE RSA KEY + IMPORT

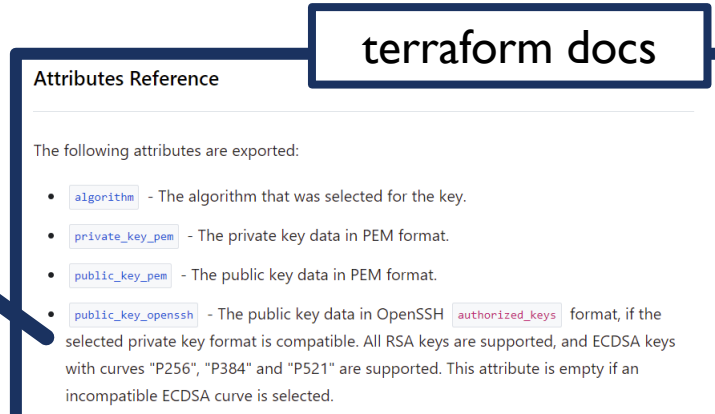
```
# Generating a random key for the project
resource "tls_private_key" "rsakey" {
    algorithm = "RSA"
    rsa_bits = 4096
}

# Import key into AWS
resource "aws_key_pair" "awskey" {
    key_name = "${var.project}-key"
    public_key = tls_private_key.rsakey.public_key_openssh
}
```

resource reference

dependency

terraform docs



Attributes Reference

The following attributes are exported:

- `algorithm` - The algorithm that was selected for the key.
- `private_key_pem` - The private key data in PEM format.
- `public_key_pem` - The public key data in PEM format.
- `public_key_openssh` - The public key data in OpenSSH `authorized_keys` format, if the selected private key format is compatible. All RSA keys are supported, and ECDSA keys with curves "P256", "P384" and "P521" are supported. This attribute is empty if an incompatible ECDSA curve is selected.

EXAMPLE: GENERATE RSA KEY + IMPORT

```
# Generating a random key for the project
resource "tls_private_key" "rsakey" {
    algorithm = "RSA"
    rsa_bits = 4096
}

# Import key into AWS
resource "aws_key_pair" "awskey" {
    key_name = "${var.project}-key"
    public_key = tls_private_key.rsakey.public_key_oper
}
```

from AWS module

terraform.tf

```
terraform {
    required_providers {
        aws = {
            source = "hashicorp/aws"
        }
```

terraform docs

Resource: aws_key_pair

Provides an EC2 key pair resource. A key pair is used to control login access to EC2 instances.

Currently this resource requires an existing user-supplied key pair. This key pair's public key will be registered with AWS to allow logging-in to EC2 instances.

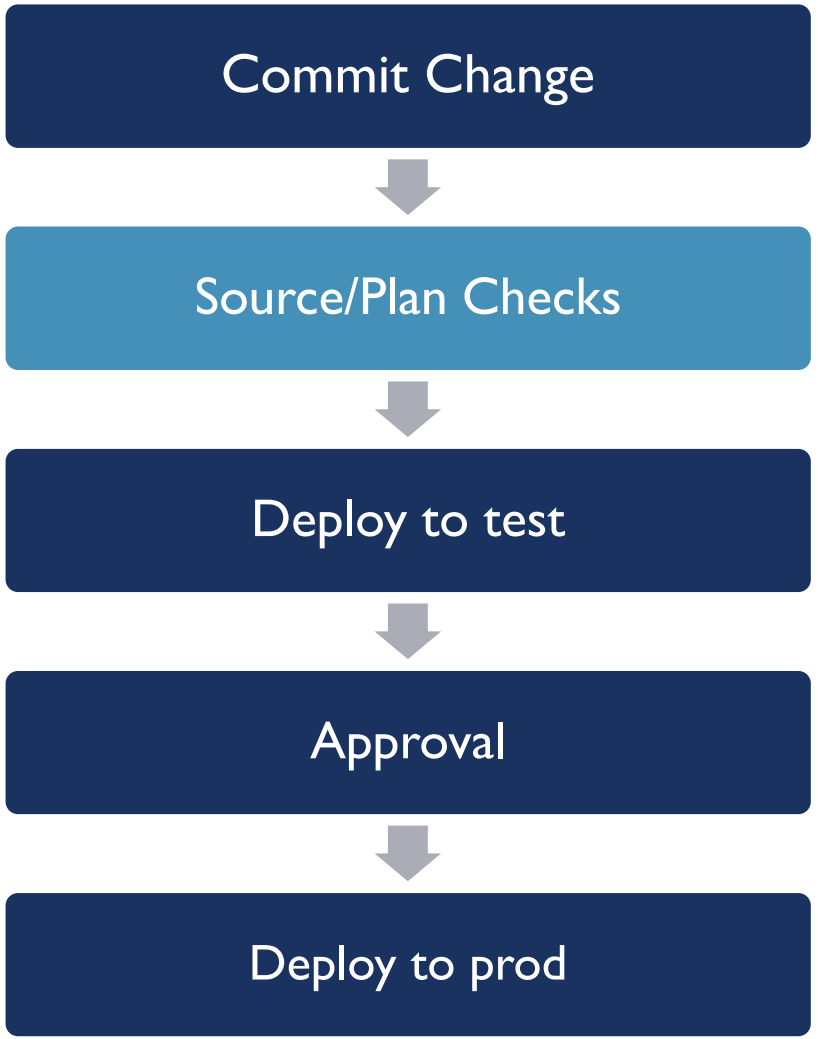
When importing an existing key pair the public key material may be in any format supported by AWS. Supported formats (per the AWS documentation) are:

- OpenSSH public key format (the format in ~/.ssh/authorized_keys)
- Base64 encoded DER format
- SSH public key file format as specified in RFC4716

SHIFTING LEFT

- Premise: catch security issues prior to deployment
- Method: deploy infrastructure changes as code; catch issues by evaluating infrastructure code
- Example: https://blog.christophetd.fr/shifting-cloud-security-left-scanning-infrastructure-as-code-for-security-issues/

SHIFTING LEFT SAMPLE WORKFLOW



SHIFTING LEFT EXAMPLE: EXPOSED SSH

- Manage AWS security groups via IAC (Terraform)
- Spot below issue via CI/CD pre-deployment check (e.g. Checkov)
- Prevent deploy until change passes

```
resource "aws_security_group" "firewall_sg" {
  name        = "firewall_sg"

  ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
```

```
resource "aws_security_group" "firewall_sg" {
  name        = "firewall_sg"

  ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["1.2.3.4/32"]
```

AREAS FOR ABUSE

- Terraform is code execution
- Main concern
  - how its handled in CI
  - who can access CI
  - who can write to source

SECRETS MANAGEMENT

- Secrets in source
- Use secret vault

- Secrets in state files
- Strong access controls on remote state

terraform docs

- Terraform Cloud always encrypts state at rest and protects it with TLS in transit. Terraform Cloud also knows the identity of the user requesting state and maintains a history of state changes. This can be used to control access and track activity. Terraform Enterprise also supports detailed audit logging.

DEPENDENCY MANAGEMENT

- Code exec, exfiltration

- Trusted modules/providers
- Explicit modules directory
- Dependency pinning

TERRAFORM FEATURES

- Local exec
- External

- Be wary
- Avoid influenceable input in command

TERRAFORM PLAN

- Treat plan as apply
  - https://alex.kaskaso.li/post/terraform-plan-rce
- Explicit modules directory

QUESTIONS?