Stephen Woodrow / @srwoodrow
SANS Cloud INsecurity Summit / 8 & 11 June 2018

Cloud security at Lyft

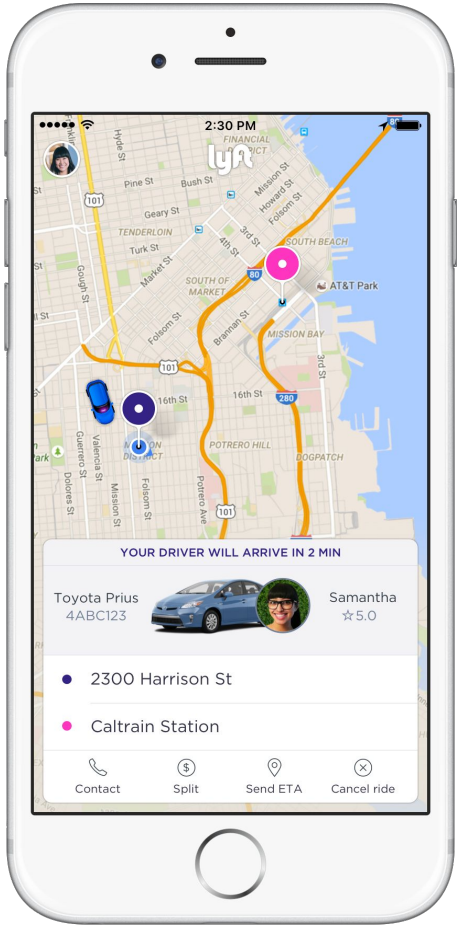
Agenda

- **Overview: Lyft & our cloud environment**
- **Making cloud security happen at Lyft**
  - Service organization
  - Resource orchestration
  - Identity & access controls
- **Cloud-native security tactics**
- **Q&A**

Overview: Lyft & our cloud environment

What is Lyft?

- Lyft is a rideshare service operating in the US and Canada
- Started as a hackathon project in 2012, the Lyft service has grown very rapidly: we now serve over one million rides/day
- From a tech standpoint:
  - Lyft is *cloud-native*—we have hosted our backend services in AWS since the first Lyft ride
  - Our engineering org is ~500 software engineers and many more tech users/consumers
  - We have a microservices architecture and operate hundreds of services on thousands of EC2 instances



Lyft’s engineering culture

- ***“Make it happen”*** is one of three core values at Lyft
- Engineers are empowered to—and accountable for—making it happen:
  - Devops model for service ownership, deployment, and maintenance
  - Heavy automation supporting SDL processes, CI/CD, monitoring, etc.
  - Few change management checkpoints with human gatekeepers
- Engineers making it happen: 200+ deploys/day

Making cloud security happen at Lyft

Organizing cloud resources

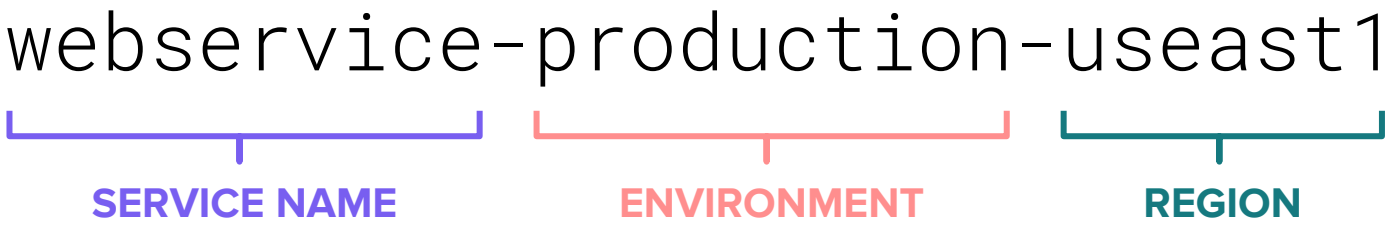
- At the scale of thousands of instances and millions of cloud resources, we need abstractions to help stay organized and reason about security policies
- At Lyft we organize a number of the primitives AWS offers us into a rough abstraction we consider a service to help ***“make it happen”:***
  - Single application deployed per service
  - Default access to resources inside service boundary
  - Default isolation from other services and resources outside service boundary

Service naming at Lyft

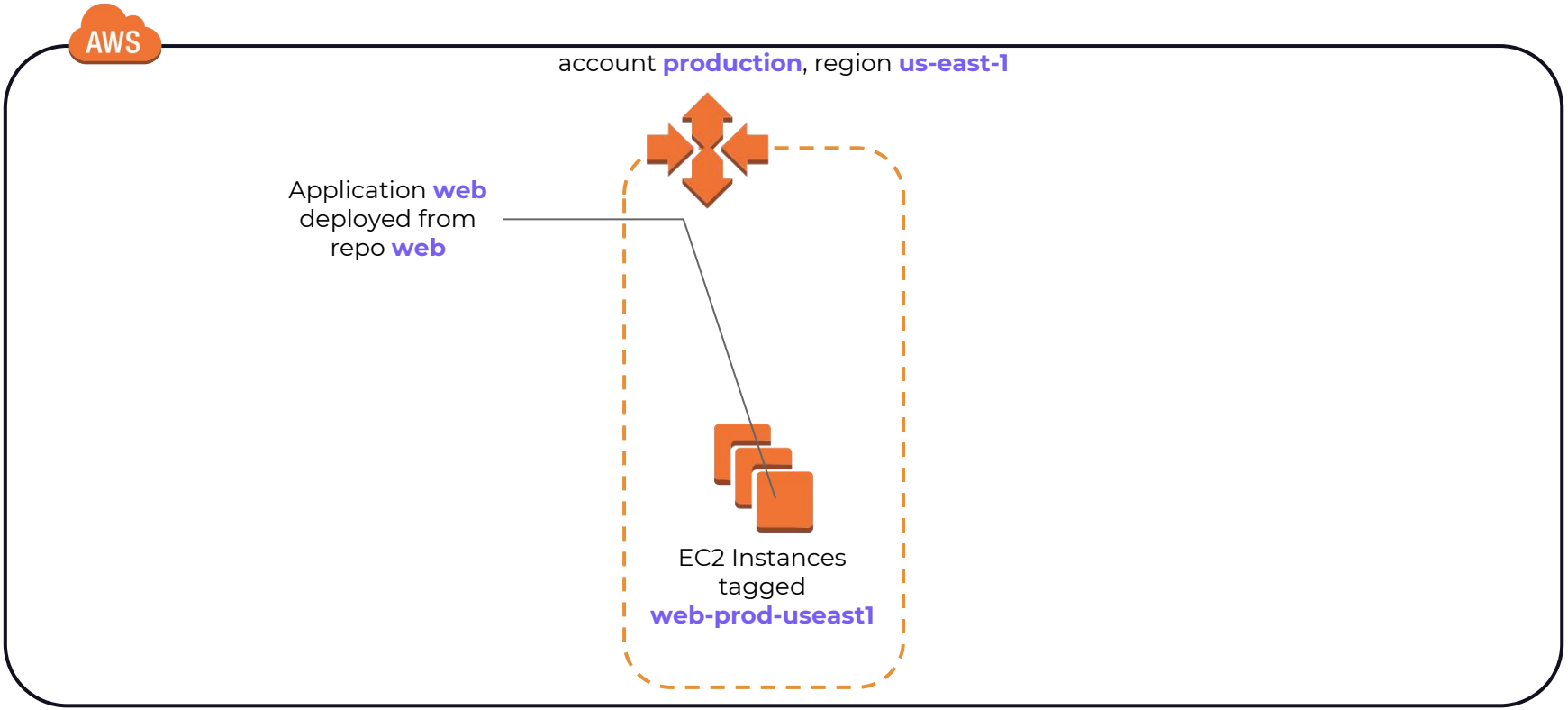
Service naming at Lyft

`webservice-production-useast1`

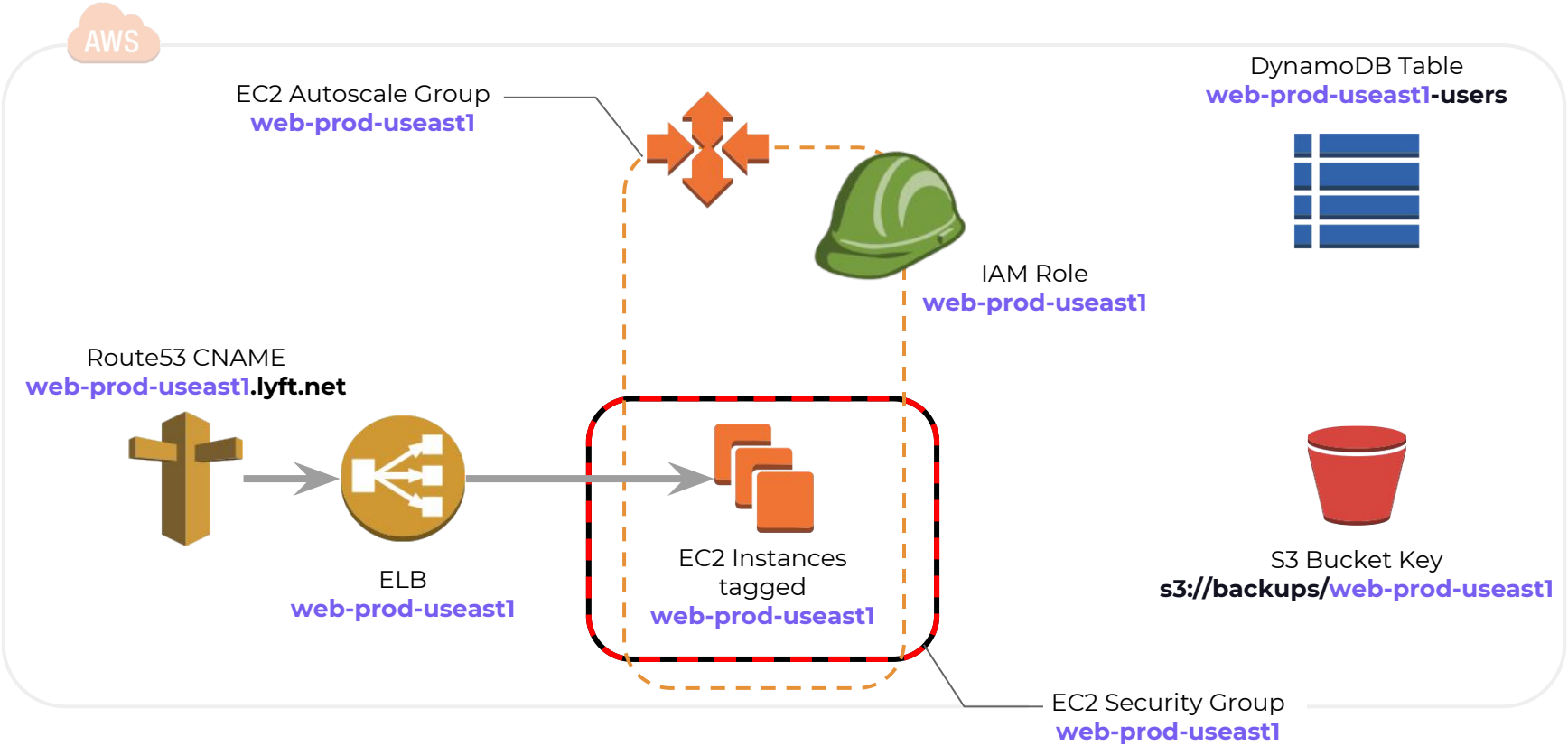
Service naming at Lyft



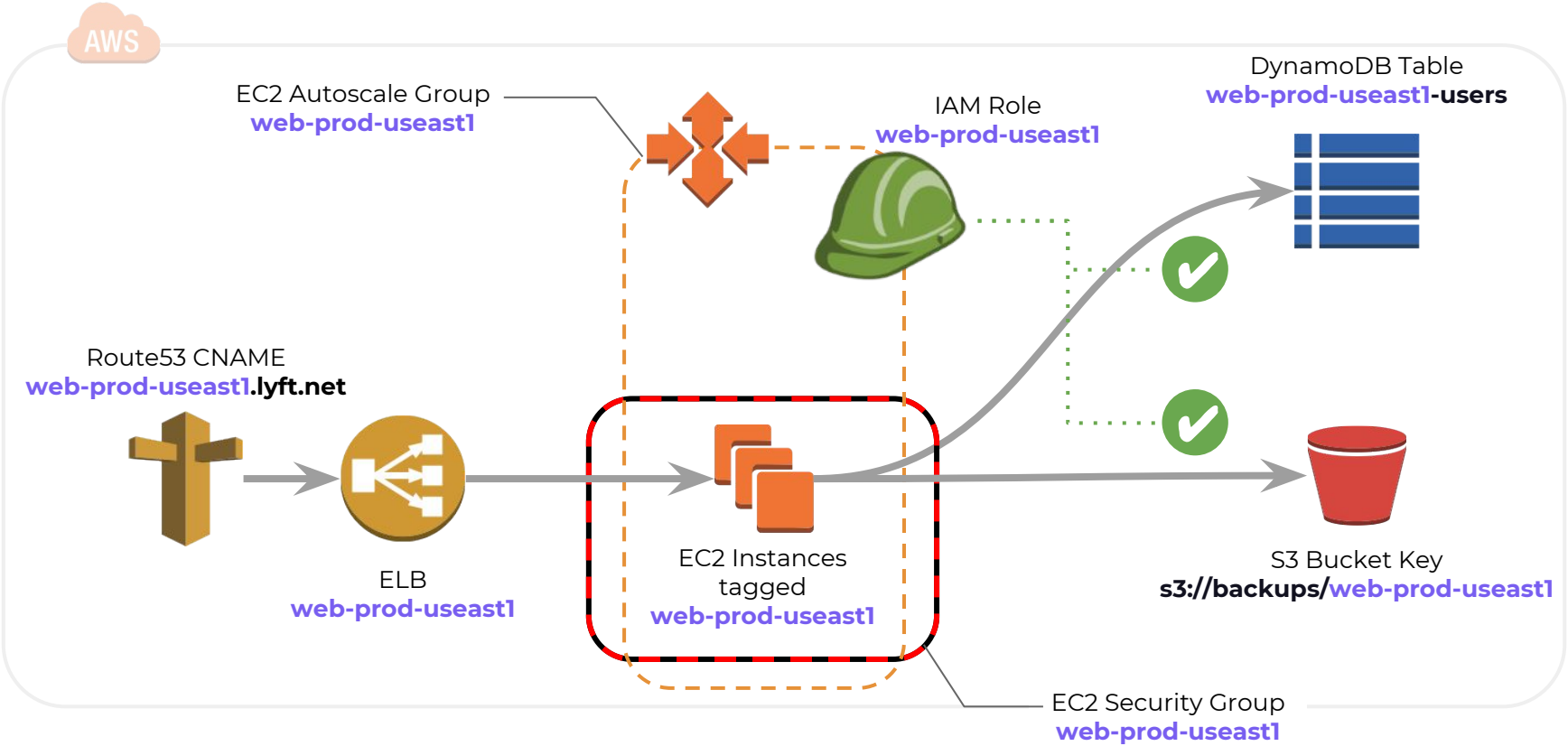
Service organization at Lyft



Service organization at Lyft



Service organization at Lyft



Lessons learned: service organization

- Standardizing service and resource naming makes many things easier:
  - Ownership, inventory, accounting
  - Creating a common mental model, making your docs higher-leverage
  - Templating and automation for service creation and maintenance
- Default IAM policy maintains strong isolation & protection
- Larger/complex services may need internal segmentation (or decomposition into smaller services) to achieve desired security properties

Cloud resource orchestration

Cloud resource orchestration

- Orchestration lets us define infrastructure with code, enabling:
  - Repeatable workflow for making changes—no console or laptop changes
  - Code review & automated testing of infrastructure changes
  - Code repo as source of intent for analysis, incident response, etc.
- Enabling ***“make it happen”:***
  - Service-specific resources are self-service and deployed with service repository
  - High-risk or account-/region-wide resources and default values are managed in a central repository

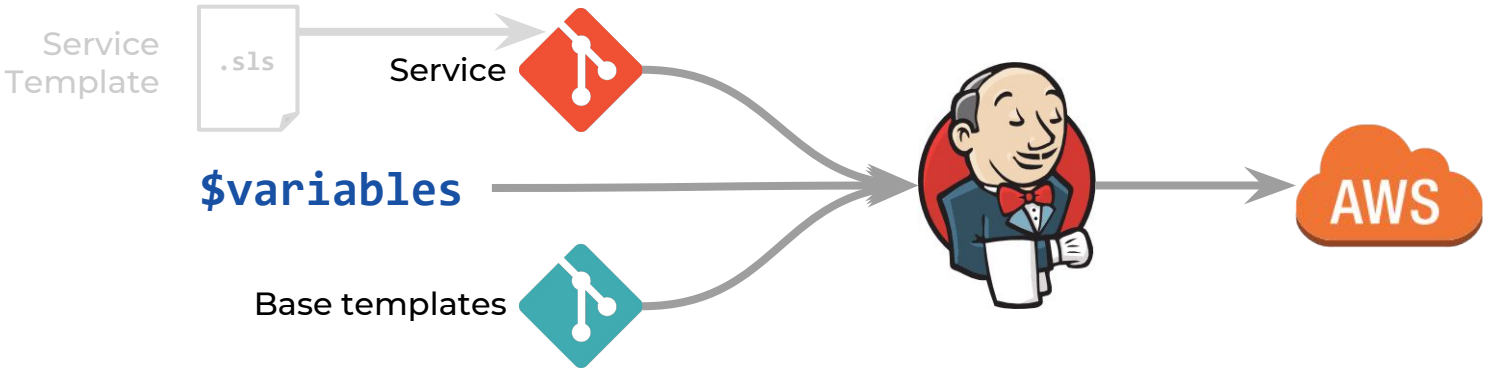
Templated self-service orchestration

- Lyft uses Saltstack* for AWS orchestration
- Service templates are used to generate basic resource manifests for new services
- Resource names and policies based on service-specific variables (e.g. service name) allow creation of service-isolated sets of resources

* terraform or cloudformation are better choices for new projects

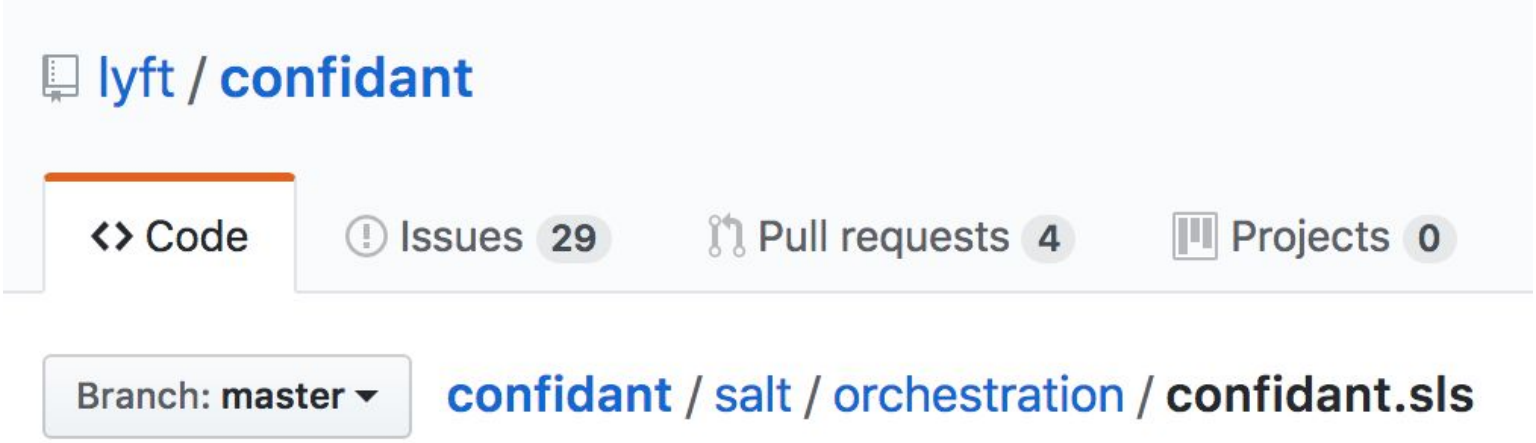
Templated self-service orchestration

- Lyft uses Saltstack* for AWS orchestration
- Service templates are used to generate basic resource manifests for new services
- Resource names and policies based on service-specific variables (e.g. service name) allow creation of service-isolated sets of resources



* terraform or cloudformation are better choices for new projects

Templated self-service orchestration



https://github.com/lyft/confidant/

Templated self-service orchestration

```
Ensure {{ grains.cluster_name }} iam role exists:
  boto_iam_role.present:
    - name: {{ grains.cluster_name }}
    - policies:
        'iam':
          Version: '2012-10-17'
          Statement:
            - Action:
                - 'iam:ListRoles'
                - 'iam:GetRole'
              Effect: 'Allow'
              Resource: '*'
        'dynamodb':
          Version: '2012-10-17'
          Statement:
            - Action:
                - 'dynamodb:*'
              Effect: 'Allow'
              Resource:
                - 'arn:aws:dynamodb:*:*:table/{{ grains.cluster_name }}'
                - 'arn:aws:dynamodb:*:*:table/{{ grains.cluster_name }}/*'
```

https://github.com/lyft/confidant/blob/master/salt/orchestration/confidant.sls

Templated self-service orchestration

```
Ensure {{ grains.cluster_name }} iam role exists:
  boto_iam_role.present:
    - name: {{ grains.cluster_name }}
    - policies:
        'iam':
          Version: '2012-10-17'
          Statement:
            - Action:
                - 'iam:ListRoles'
                - 'iam:GetRole'
              Effect: 'Allow'
              Resource: '*'
        'dynamodb':
          Version: '2012-10-17'
          Statement:
            - Action:
                - 'dynamodb:*'
              Effect: 'Allow'
              Resource:
                - 'arn:aws:dynamodb:*:*:table/{{ grains.cluster_name }}'
                - 'arn:aws:dynamodb:*:*:table/{{ grains.cluster_name }}/*'
```

https://github.com/lyft/confidant/blob/master/salt/orchestration/confidant.sls

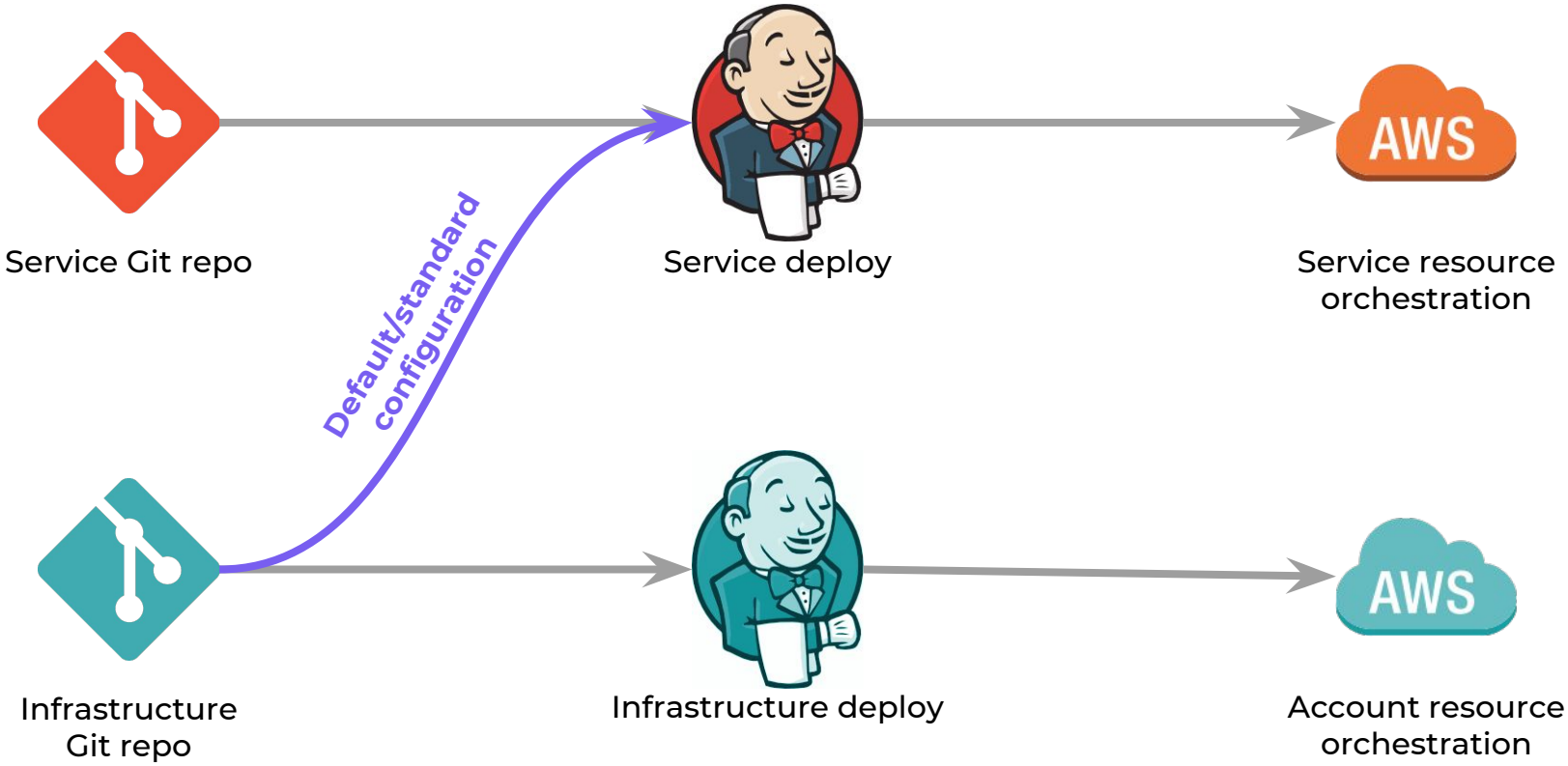
Templated self-service orchestration

```
Ensure {{ grains.cluster_name }} iam role exists:
  boto_iam_role.present:
    - name: {{ grains.cluster_name }}
    - policies:
        'iam':
          Version: '2012-10-17'
          Statement:
            - Action:
                - 'iam:ListRoles'
                - 'iam:GetRole'
              Effect: 'Allow'
              Resource: '*'
        'dynamodb':
          Version: '2012-10-17'
          Statement:
            - Action:
                - 'dynamodb:*'
              Effect: 'Allow'
              Resource:
                - 'arn:aws:dynamodb:*:*:table/{{ grains.cluster_name }}'
                - 'arn:aws:dynamodb:*:*:table/{{ grains.cluster_name }}/*'
```

confidant-production-useast1

https://github.com/lyft/confidant/blob/master/salt/orchestration/confidant.sls

Self-service orchestration



Lessons learned: orchestration

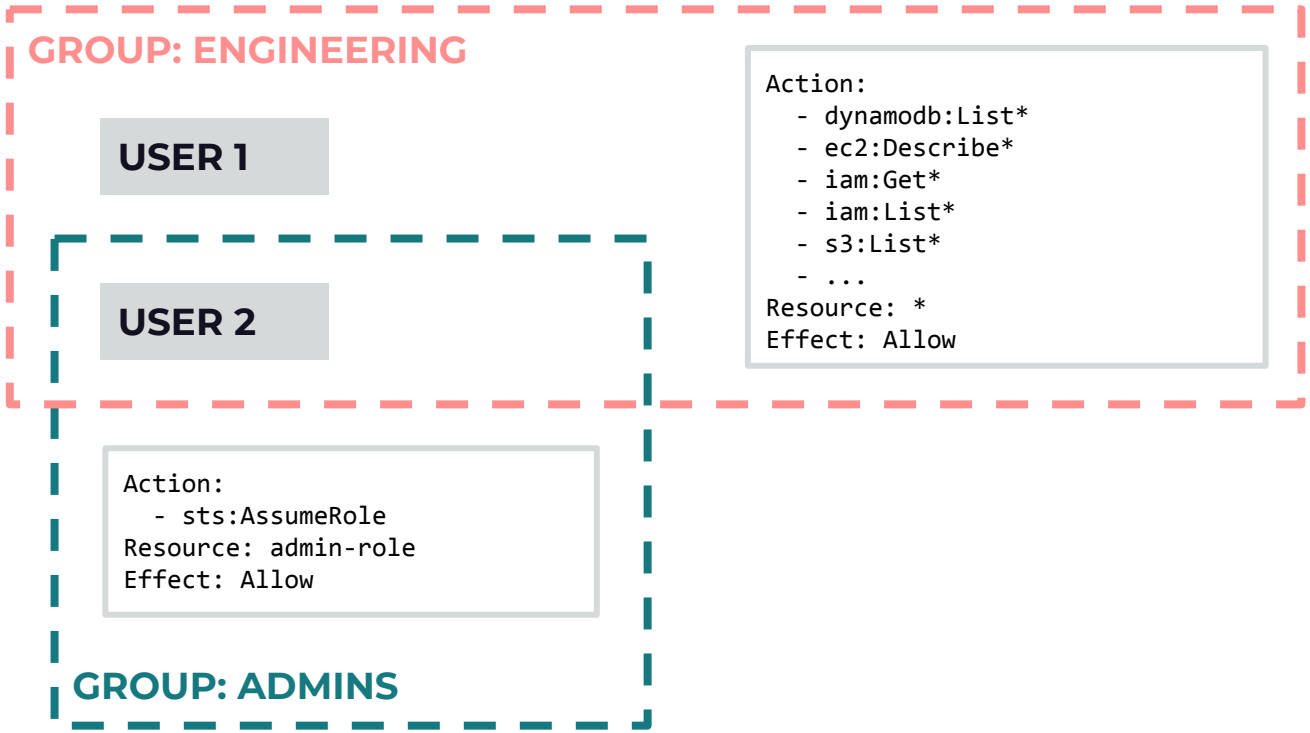
- Challenges
  - Fleet-wide changes (e.g. instance type upgrade) requires fleet-wide redeploy
  - Fine-grained resource management is probably not the right level of abstraction for most development teams
  - Automated lint/static analysis to make sure orchestration changes are safe
- Orchestration deployment tools require high-privilege IAM role
  - Jenkins become high-risk, large blast radius infrastructure
  - How do you know your tests aren't running with *:* IAM role?

Identity and access controls

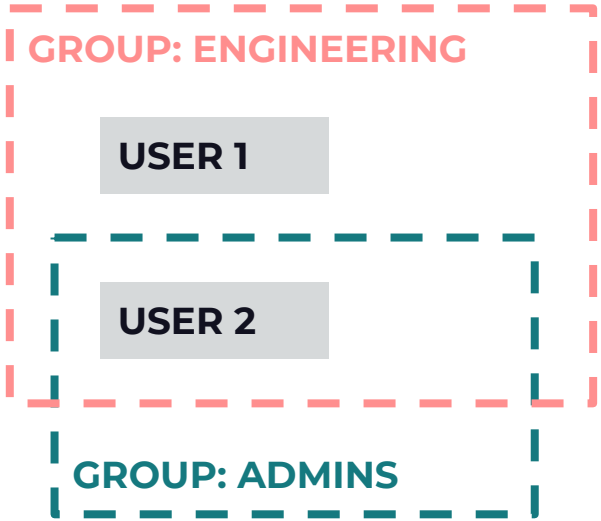
Identity and access controls (for humans)

- AWS IAM has account-wide blast radius!
- Choose the strongest, best-managed tool you've got for managing IAM Users, Roles, and Policies
  - IAM Users + orchestration vs. SSO + roles
- Enabling ***"make it happen":***
  - Self-service credential management: coinbase/self-service-iam
  - Allow engineers to list resources and elevate privileges for common ops tasks
  - Higher-risk and administrative access restricted

IAM Users & Groups: “just enough” by default



IAM Roles enable temporary elevated privilege



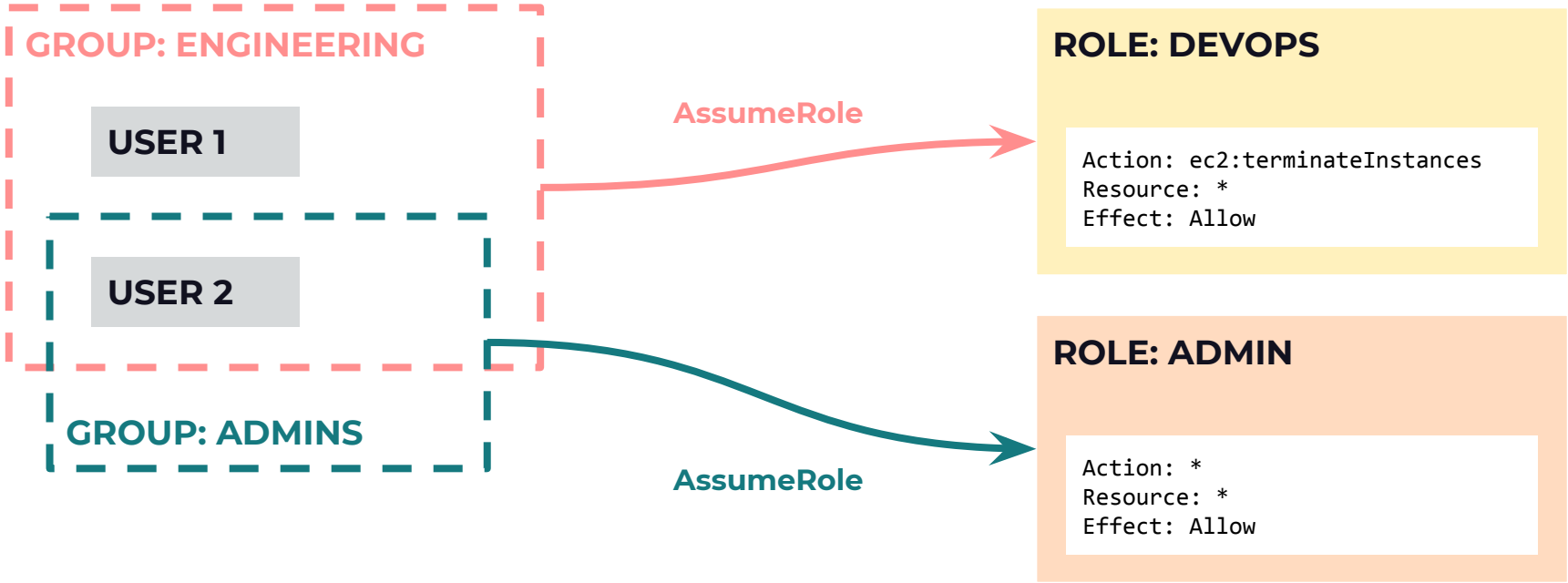
ROLE: DEVOPS

```
Action: ec2:terminateInstances
Resource: *
Effect: Allow
```

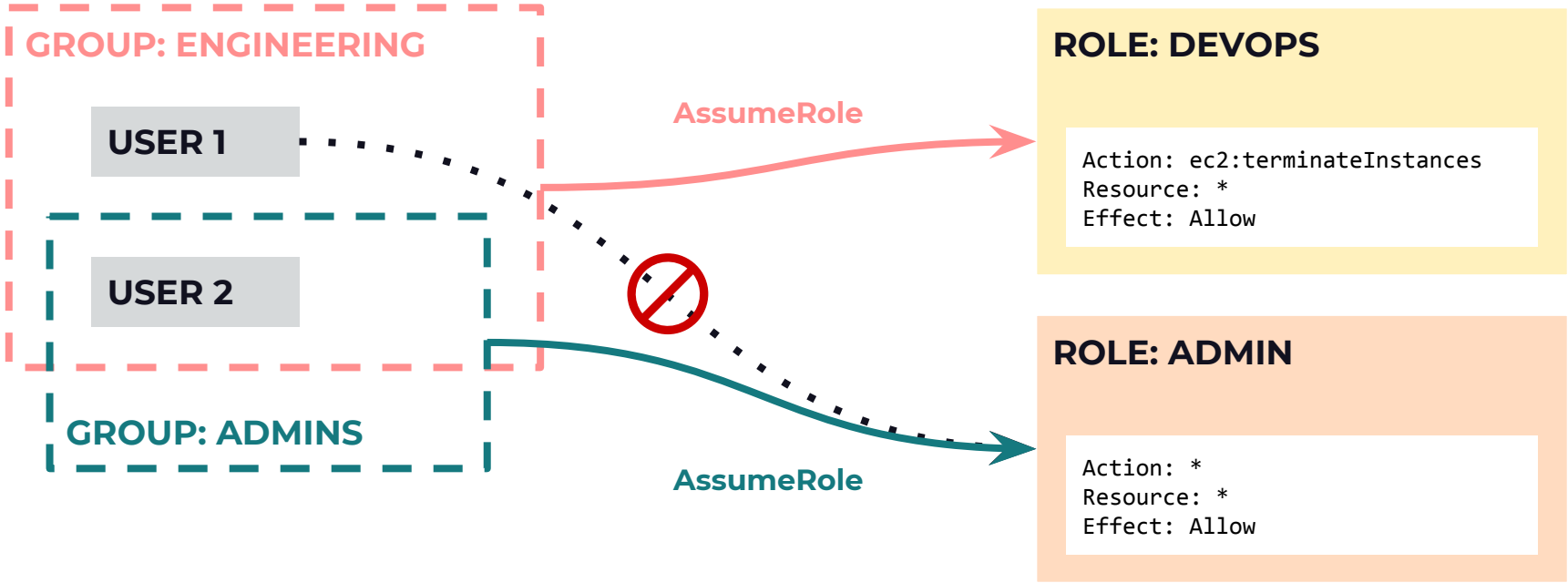
ROLE: ADMIN

```
Action: *
Resource: *
Effect: Allow
```

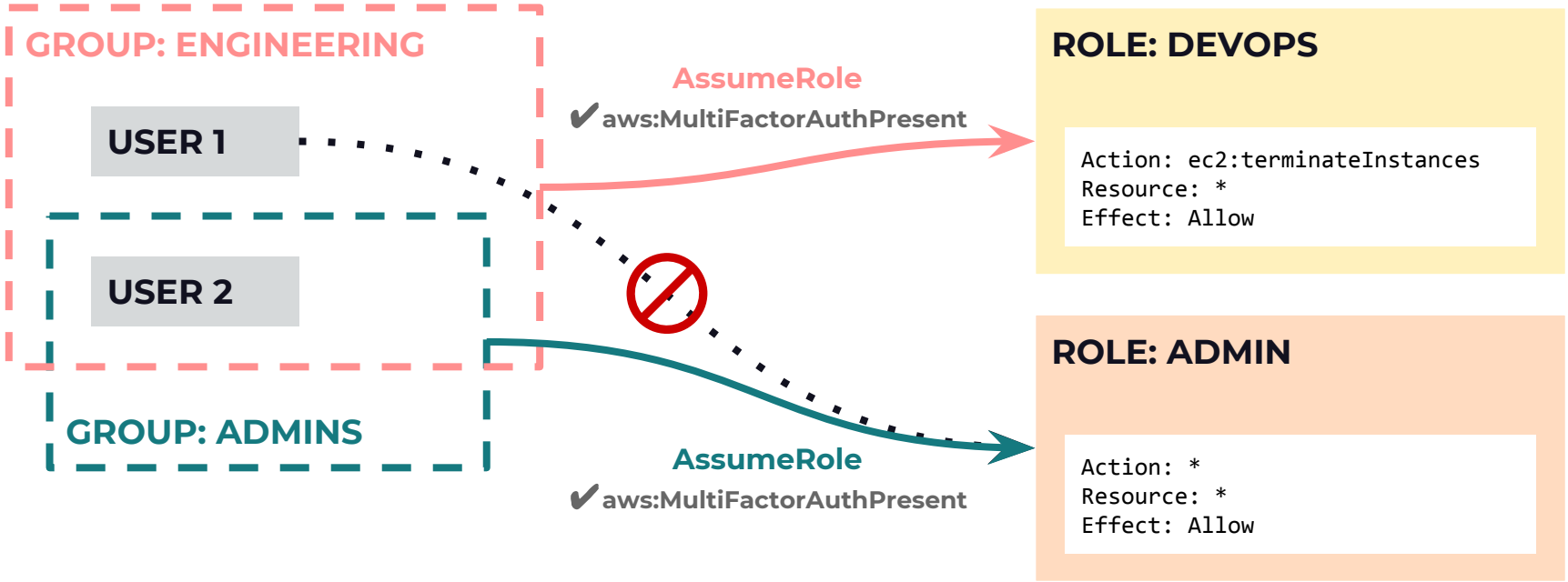
IAM Roles enable temporary elevated privilege



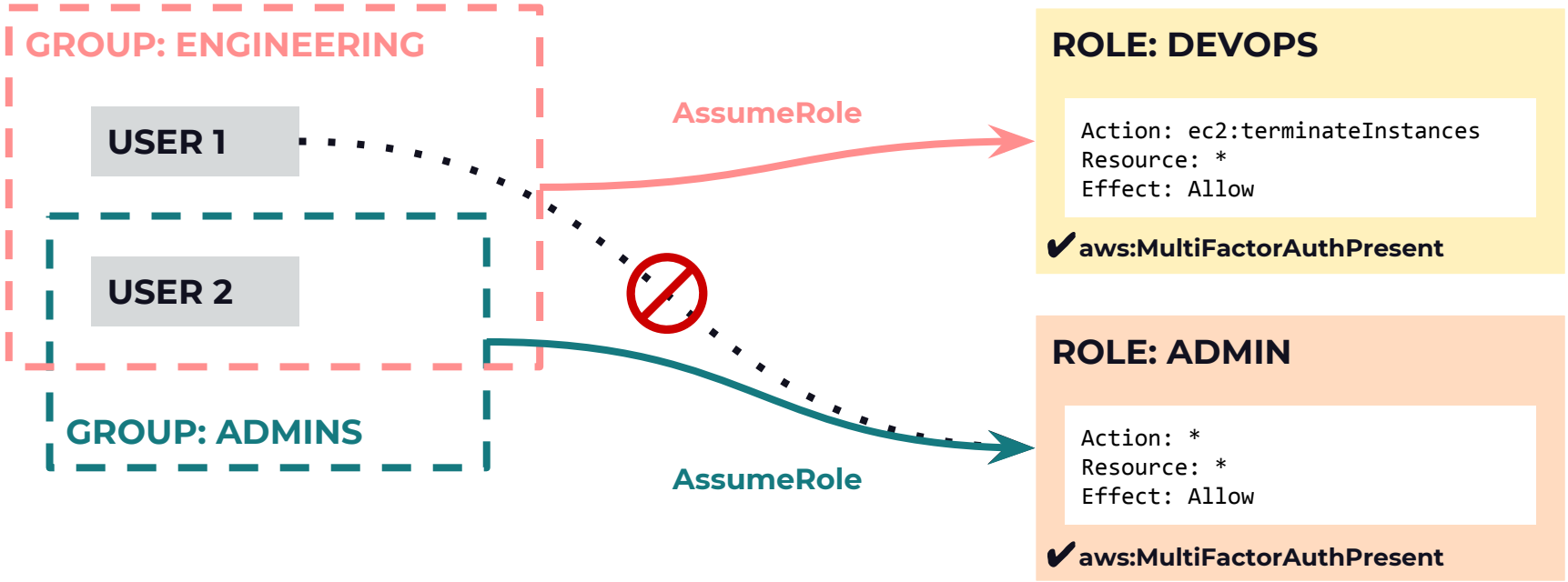
IAM Roles enable temporary elevated privilege



IAM policy to enforce MFA everywhere



IAM policy to enforce MFA everywhere



http://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.htm

Locking down the AWS root user

- The root user of an AWS account cannot be constrained
  - Don’t use except when absolutely required (pen testing, billing changes, etc.)
- MFA is a must
- No credentials issued
- Alert on any use

Identity and access controls (for machines)

- Use IAM Roles everywhere—let AWS do the hard work to make this easy for you
  - Push partners to use roles with cross-account trust
- Protect the metadata service (`http://169.254.169.254`) when it matters:
  - Docker containers: Metadata proxy https://github.com/lyft/metadataproxy
  - Webhooks: SOCKS proxy https://github.com/stripe/smokescreen

Lessons learned: Identity and access controls

- IAM Users/Access Keys can quickly get messy AND have major consequences
  - Best case: critical production dependencies that are hard to change
  - Worst case: checked into source code/out on the Internet
  - Upshot: Use IAM Users only when you have no better alternative
- Have a plan for MFA enforcement and key rotation for all IAM Users
- Consider SSO for human users, at least for non-admin roles
  - Spend your time improving security, not resetting passwords

Cloud-native security tactics

Autoscaling → Autopatching

- Autoscaling ensures you always have a set number of application instances
- Leverage the ephemeral nature of Instances to automate non-critical system patching
  - Requires system update on launch or continuously-updated AMIs/LaunchConfigurations
- Autoscaling as part of daily traffic load
  - Termination policy: `OldestInstance` or `OldestLaunchConfiguration`
- “Reaper Monkey”: explicitly terminating older instances
  - Blacklisting & scheduling to deal with more critical or stateful applications

Trust no one (else's network)

- Cloud infrastructure ➜ isolated by default ➜ reduced blast radius
- Interconnecting office networks with cloud networks ➜ increased blast radius
  - Trust administration of office network
  - Increased network scope for compliance assessments/etc
- Consider running VPN terminator service inside your cloud network instead
  - Access from office = access from home = access from coffee shop

Brawn over brains

- AWS can sometimes make the easy things hard, but also makes hard things possible
- Using automation to leverage the incremental pricing and elastic nature of cloud resources can yield new solutions to old problems
  - AWS Lambda: massively parallel binary malware analysis: https://www.binaryalert.io/
  - AWS S3 + Athena: Collect all the data you want, and dig into it later only if you need to do incident response/etc.
  - AWS Organizations: create an AWS account per service/application for even greater isolation

Thank you

8 & 11 JUNE 2018