RSAConference2019
San Francisco | March 4–8 | Moscone Center

SESSION ID: STR-T08

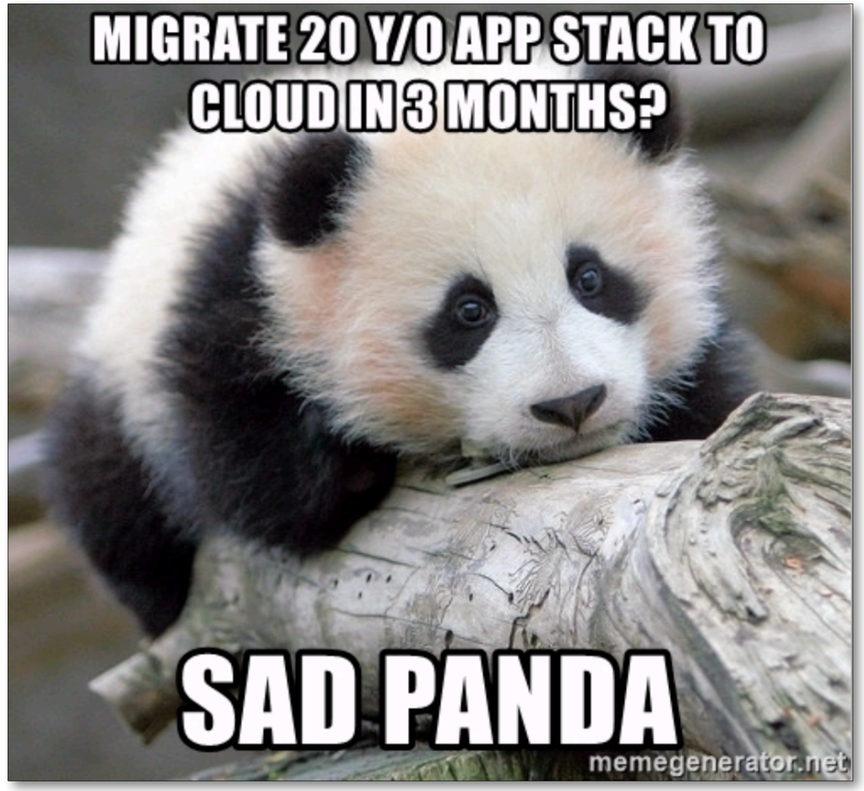
Lift and Shift, Don’t Lift and Pray: Pragmatic Cloud Migration Strategies

Rich Mogull

Analyst/CEO Securosis
VP of Product DisruptOps
rmogull@securosis.com

Reality Bites

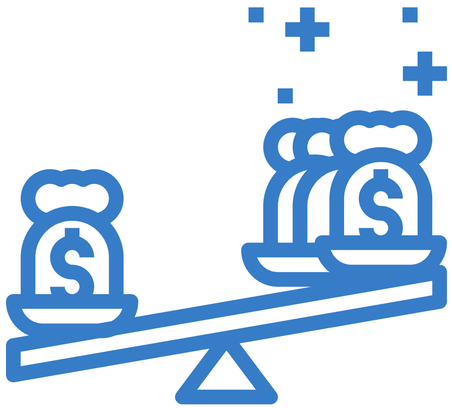
- There is relentless pressure to move to cloud
- Nearly all published best practices are designed for clean builds, not migrations
- Some of the “best practices” for migrations are really cloud antipatterns, but are pushed to make it look easy/fast



A Gilded Age of Lift and Shift

- The migration of all or some of an existing application stack(s) from an on-premise deployment to a public cloud provider.
- This does not include migration to a remote datacenter
- The application architecture itself is irrelevant (new/old/anything counts)

Why Lift and Shift?



Leverage Existing Investments

Pressure to Vacate Datacenters

Limited Resources for New Builds

Perils

Greater Costs

- Can't optimize for cloud
- Little to no elasticity
- OpEx/manual management

Less Secure

- Little or no immutable
- Tied to legacy security model
- Often manual management (no automation)

Less Reliable

- Fragile architectures
- Manual DR
- Can't leverage inherent cloud resiliency

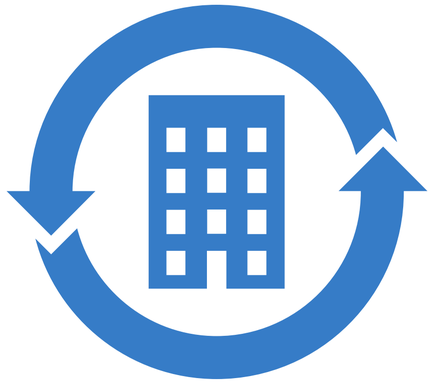
Less Agile

- Can't leverage many cloud capabilities
- Tied to manual processes for maintenance and updates

Lift and Shift Options

Rearchitect

Start from scratch
(or close to it)



Refactor

Modify what you
can and improve
over time

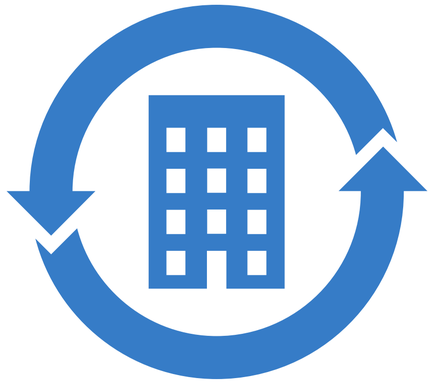
Rehost

Just move it

Lift and Shift Options

Rearchitect

Start from scratch
(or close to it)



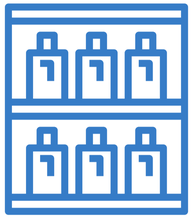
Refactor

Modify what you
can and improve
over time

(Lift and Pray)

Just move it

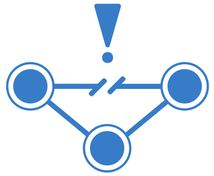
Constraints and Complexities



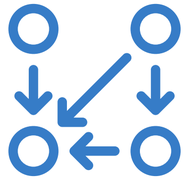
COTS

Skills and Staffing

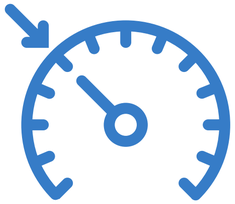
Network Constraints
(Including hybrid)



Resiliency Constraints



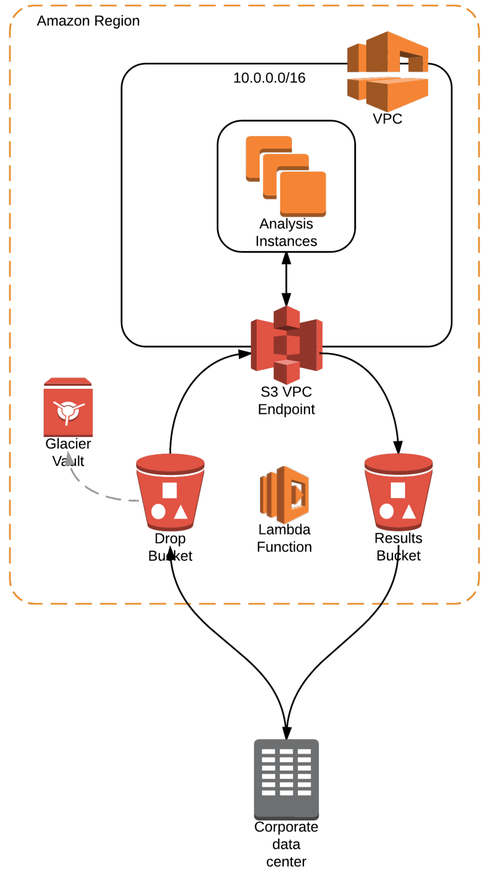
Dependencies
(e.g. databases, directory servers)



Service Limits

Rearchitect

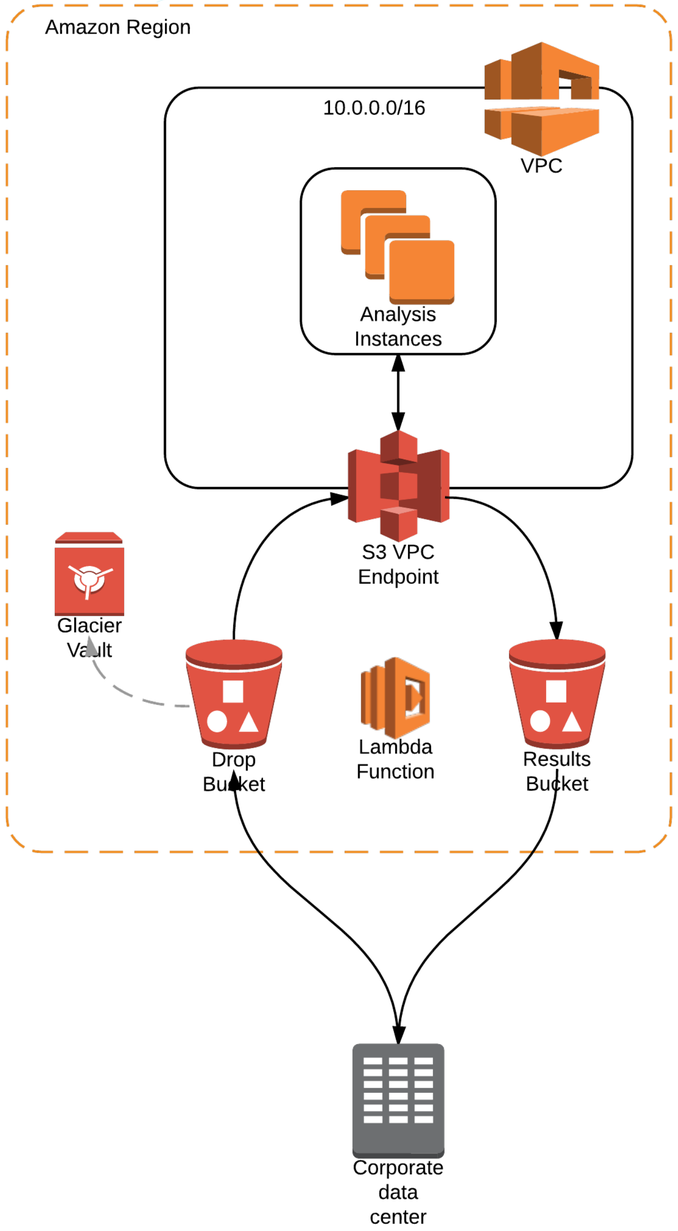
- Completely redesign the application stack
  - Or cleave off part of it for a redesign
- Implement cloud-native from scratch (if needed)
- Often done in parallel with one of the other options
- Best in the long term, but often not an option



Big Data Analytics

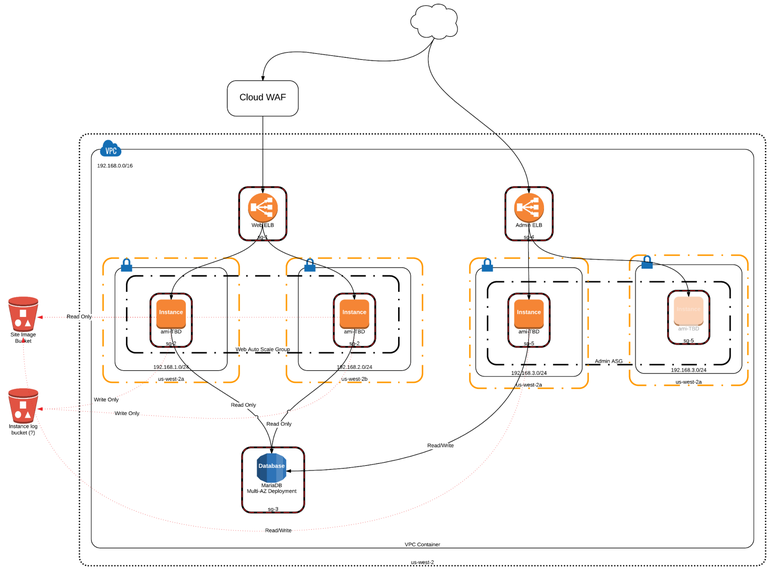
Rearchitect

- Serverless (mostly)
- Immutable
- Event-driven
- Small attack surface
- Cost-effective
- Leverages existing analytics code



Refactor

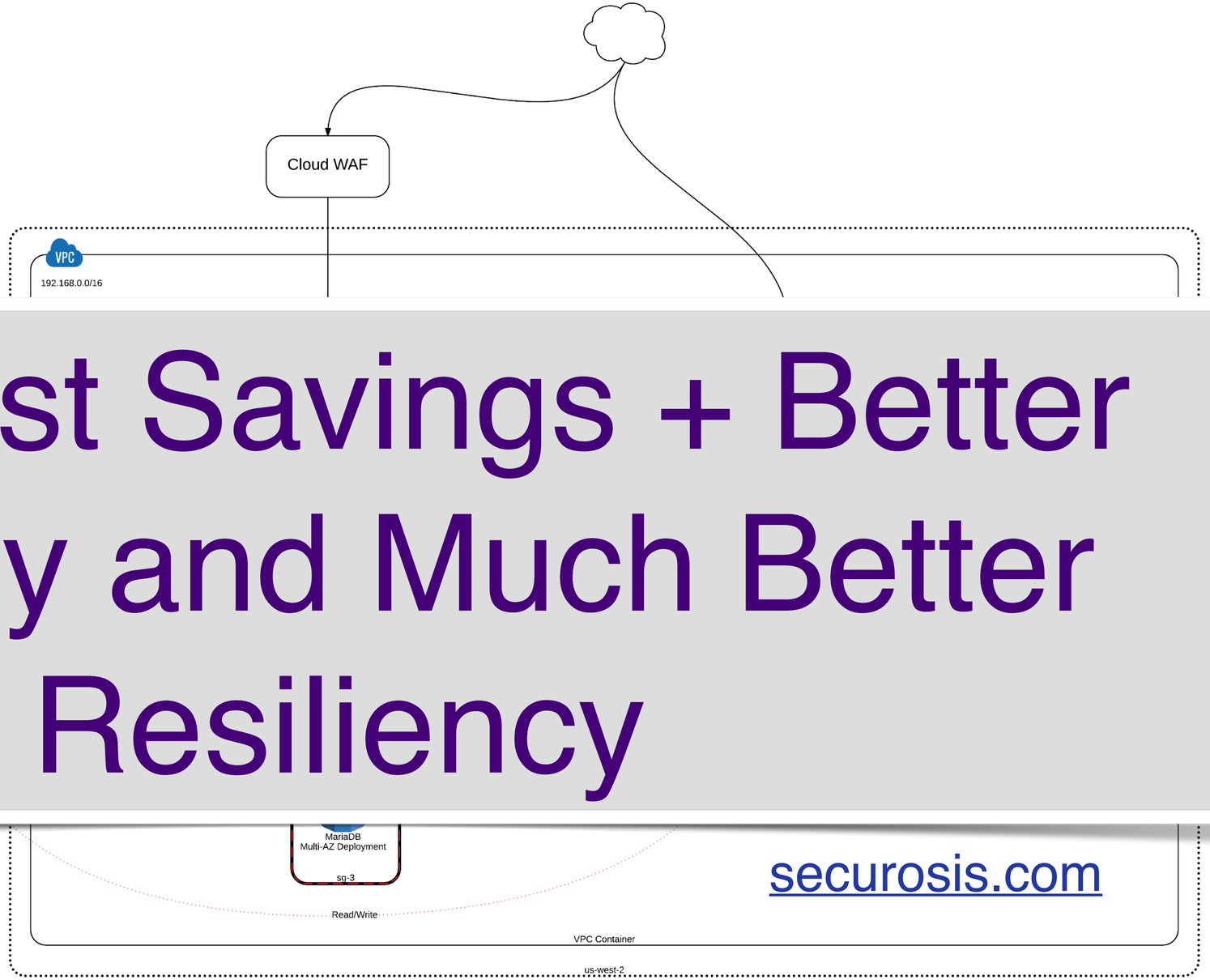
- Take the existing stack, change what you can, live with the rest
  - Use a dedicated account/subscription/project
  - Replace easy-to-swap components like load balancers
  - Leverage autoscaling if possible, but config management if not
  - (more to follow)
- You work within the existing application's constraints but go as cloud-native as possible
- Frequently the most-balanced option



COTS CMS/Website

Refactor

- Ad
  sca
- Sh
  to
- Us
- Se
- Isc

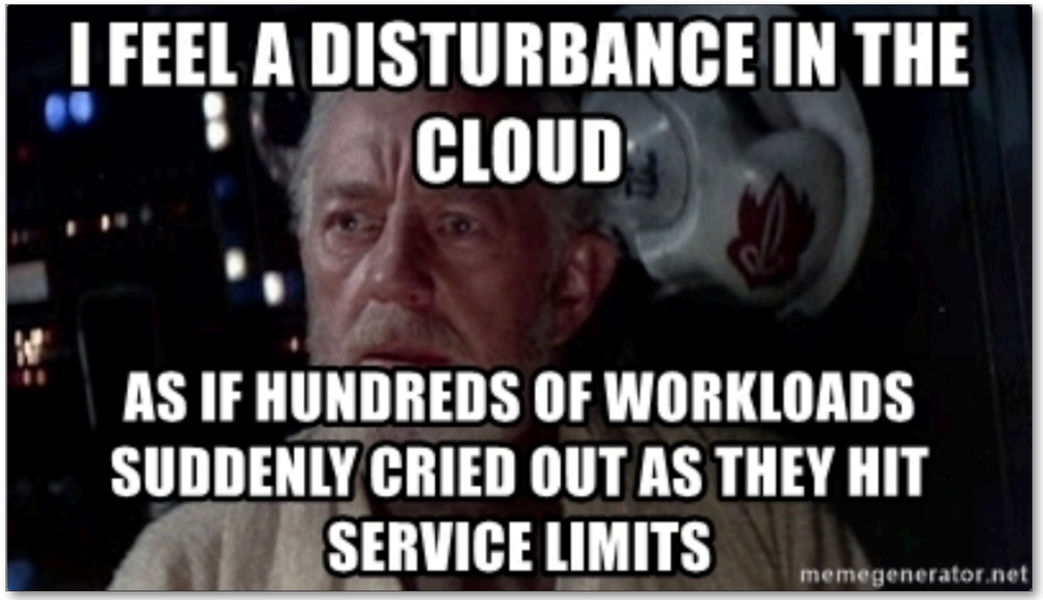


10X Cost Savings + Better Security and Much Better Resiliency

securosis.com

Rehost

- "Lift and shift" in the purest sense
  - No architectural changes
  - Typically put into a network that replicates on-prem datacenter
  - Security controls also lifted and shifted
- Most projects run into significant constraints within 2-3 years
  - IAM and service limits the biggest culprits
- Results in anti patterns being used
- Recommended all too often
  - Cloud providers just want the workloads as fast as possible
  - Many consultants love the big projects



Lift and Pray

- Network materially harder to secure
- Security tool dependencies often expand attack surface
- IAM extremely difficult to constrain
  - It's like giving everyone their own keys to the datacenter
- Brings all the old security problems
- Creates a large blast radius
- You will fail cloud-specific security audits
- Service limits will inhibit proper cloud security operations
- Incident Response harder and often not fully prepared before migration
- BCP/DR ugly and expensive

Lift and Pray

- Network materially harder to secure
- Security tool dependencies often expand attack surface
- IAM extremely difficult to constrain
  - It's like giving everyone their own keys to the datacenter
- Brings all the old security problems
- Creates a large blast radius
- You will fail cloud-specific security audits
- Service limits will inhibit proper cloud security operations
- Incident Response harder and often not fully prepared before migration
- BCP/DR ugly and expensive

How to Choose

Rearchitect	Refactor	Rehost
When you have the time	When you can't rearchitect right away	When you don't have a choice
When you have the people and skills	When you have a mix of people and skills	When you have a backup plan
When you can use it to learn and rebuild your program	When you can balance security using isolation	Or
Also a great option in parallel	When you are able to iterate over time	*When you know you have a better job in <18 months when it crashes :)*

RSAConference2019

How to securely lift and shift

The Roadmap

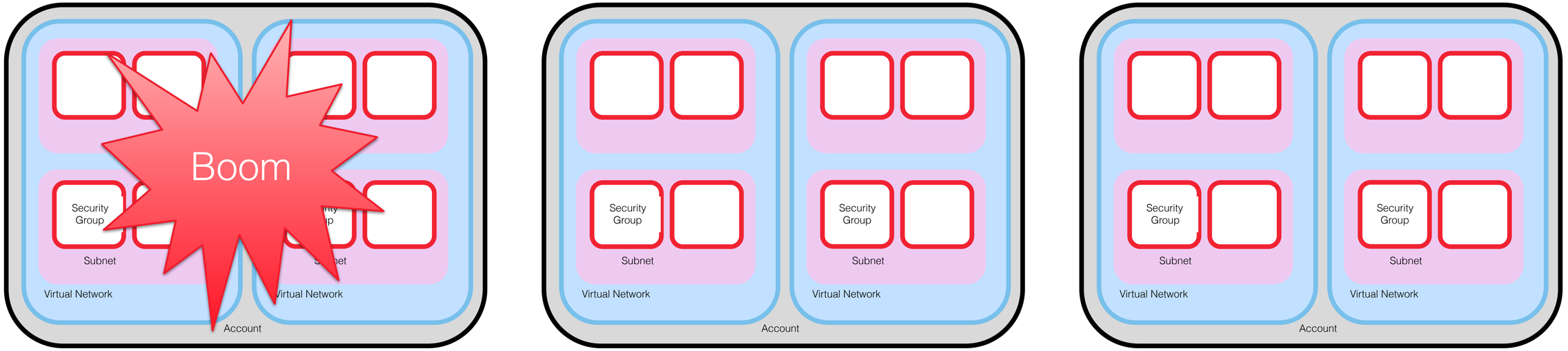
1. Analyze app dependencies and current security
2. Isolate/compartmentalize
3. Match infrastructure to the app
4. Implement true least privilege network
5. Lock down IAM
6. Turn on native security controls/tools
7. Integrate PaaS/refactor
8. Build/integrate shared security services
9. Set guardrails
10. Codify into infrastructure as code
11. Determine your expiration date

Analyze app dependencies and security

Map out app + infrastructure

- COTS
- Databases, storage, data warehouse
- OS/config
- Network connectivity
- Network filtering/IDS/IPS/WAF
- EPP/host security agents
- Logging/monitoring
- DR/BCP
- Metastructure (e.g. directory servers, IP addresses, DNS, key management)
- Privileged user access/change management/deployment pipeline (or not)
- Sessions and load balancers

Isolate/compartmentalize



Dev

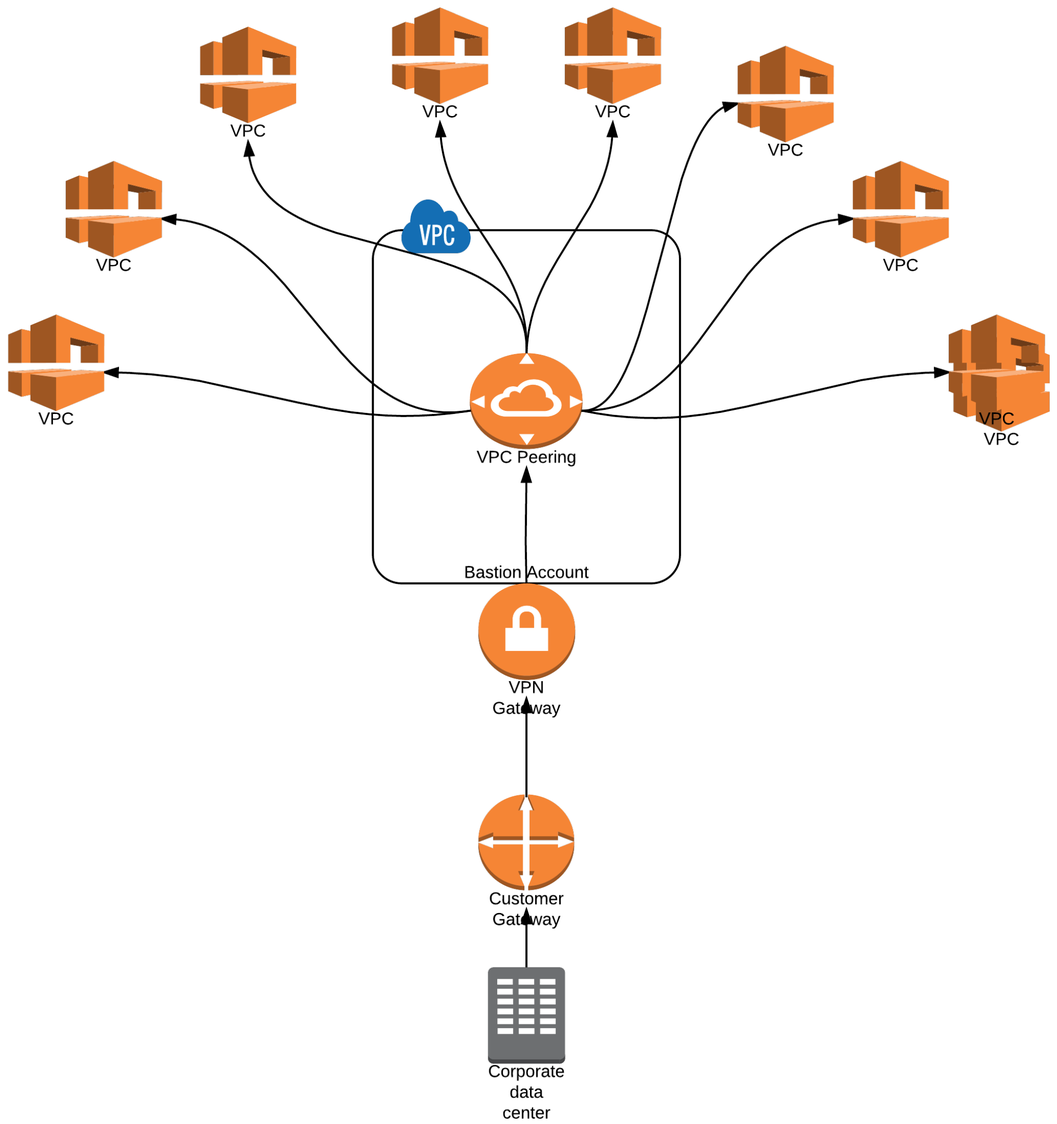
Test

Prod

2-3 accounts/projects/subscriptions per project/app stack

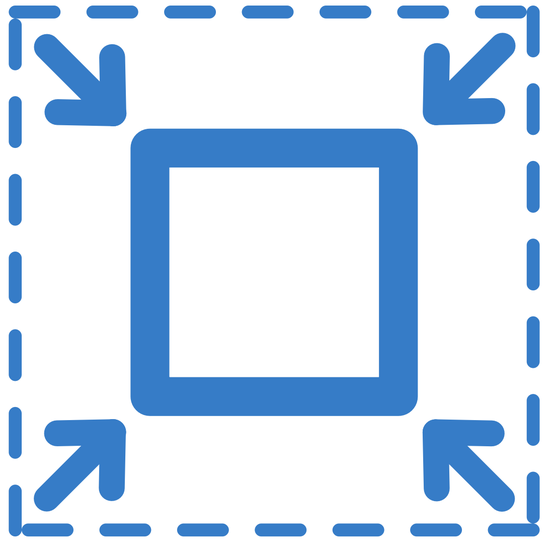
Transit Networks

Not a panacea, use only when required

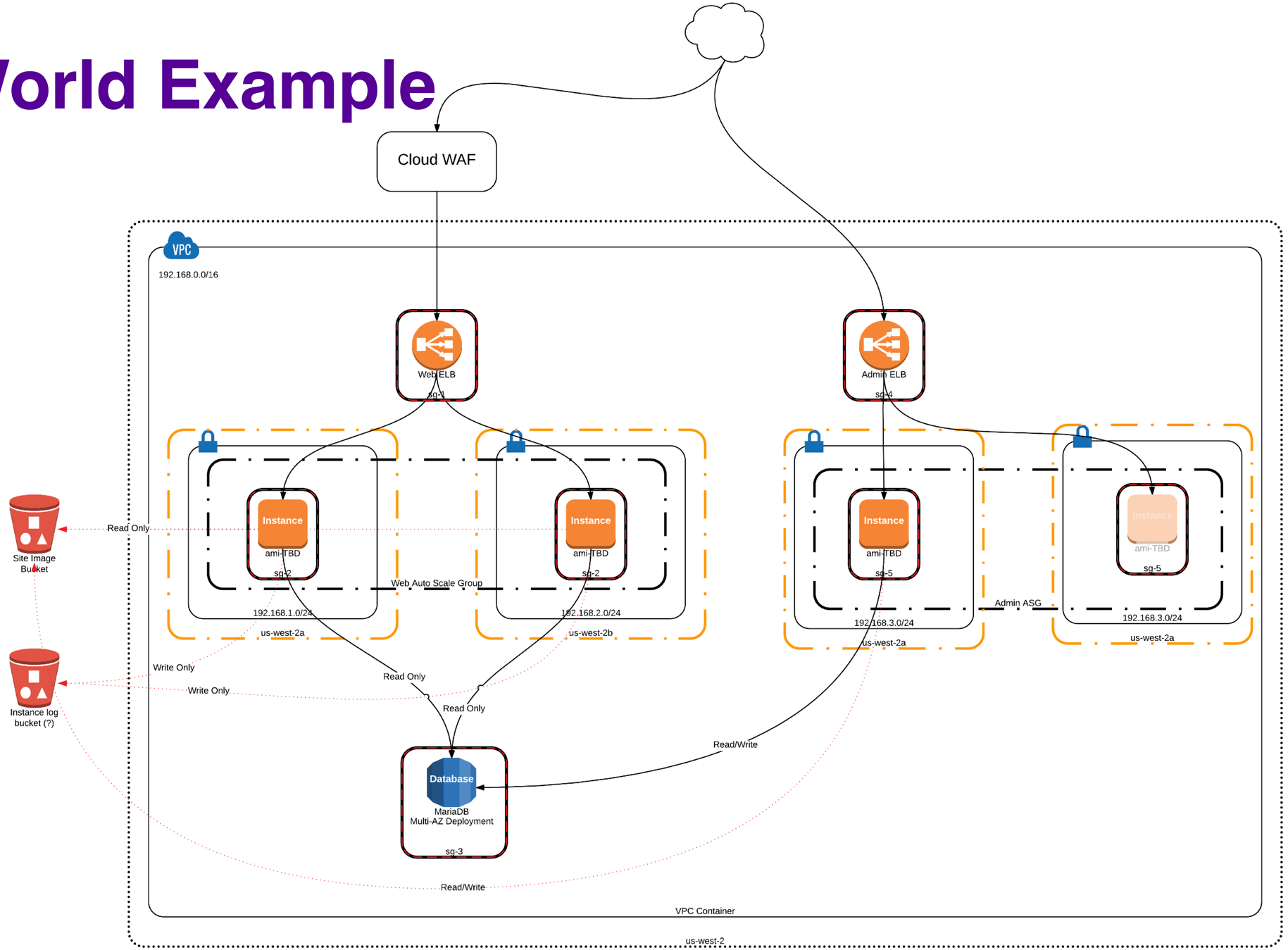


Match the infrastructure to the application

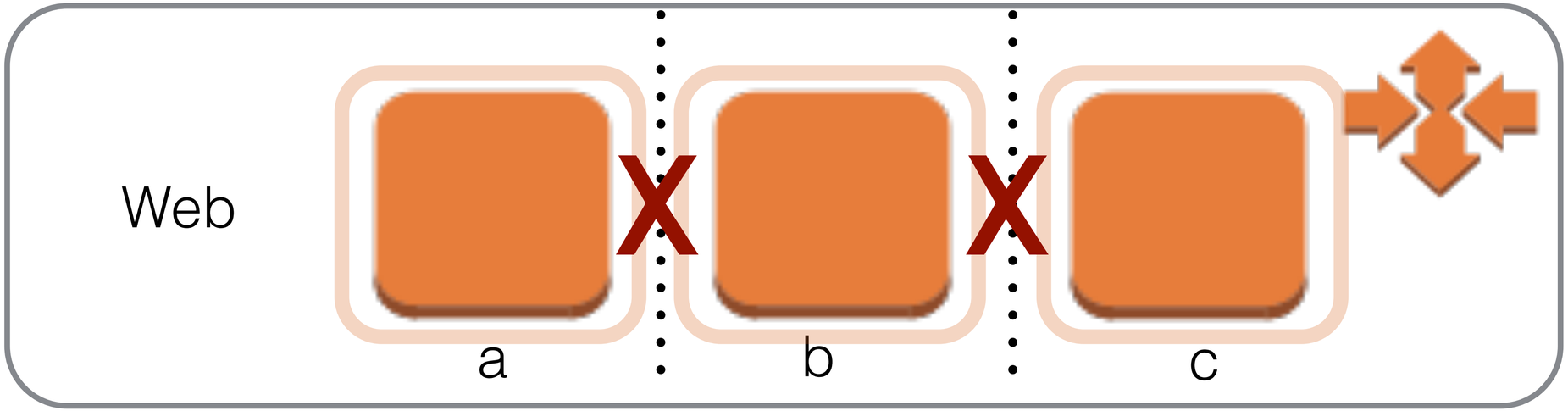
- Minimum required network
- Minimum required services
- Reduce external dependencies as much as possible
- Hybrid when required, not as a default



Real World Example



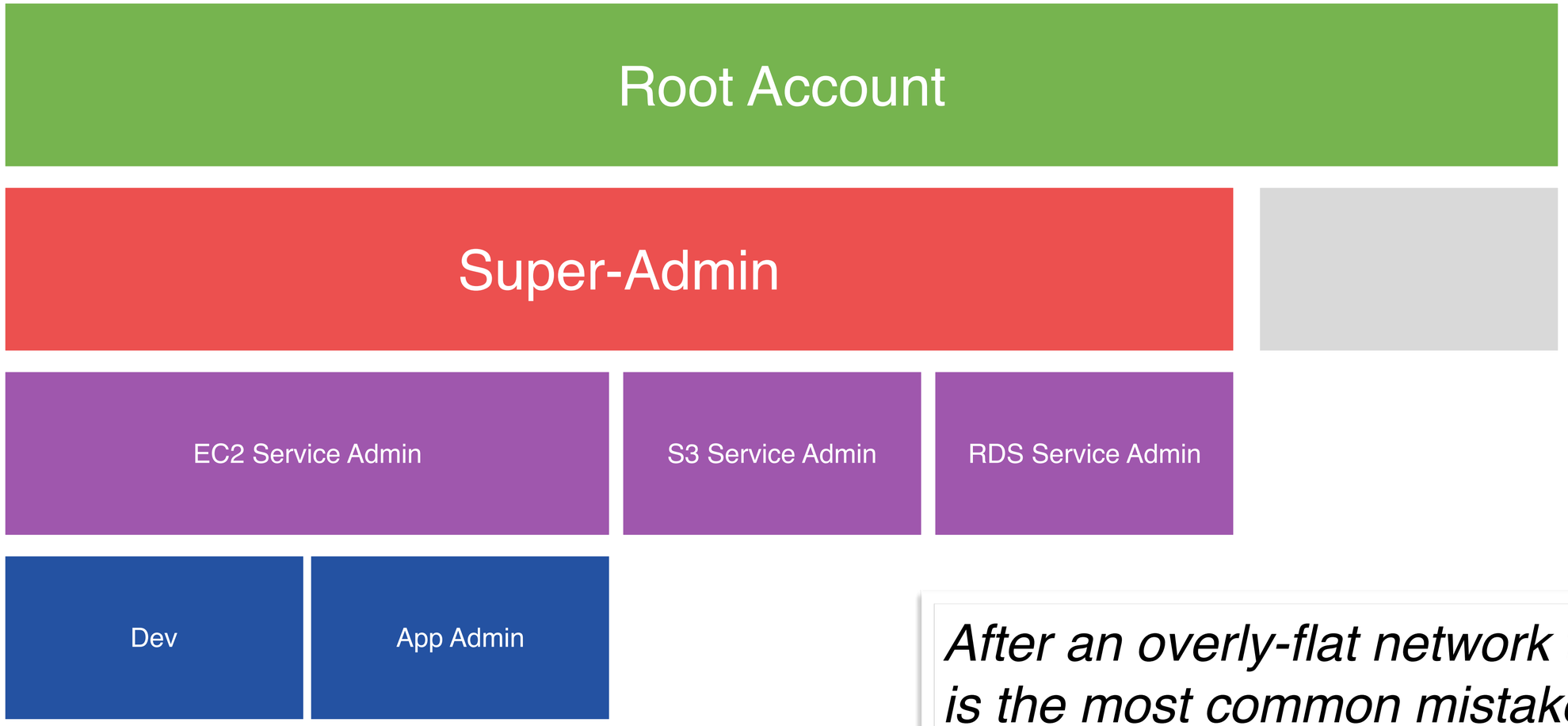
Implement a least privilege network



Type	Protocol	Port Range	Source
SSH	TCP	22	68.2.174.98/32
HTTP	TCP	80	0.0.0.0/0

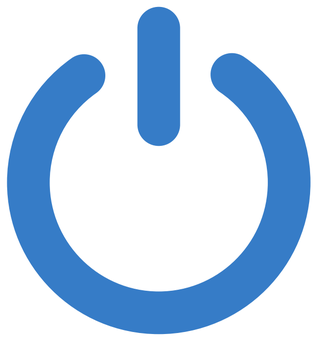
- *When assessing lift and shift projects this is unfortunately usually the exception, not the rule*

Lock down IAM



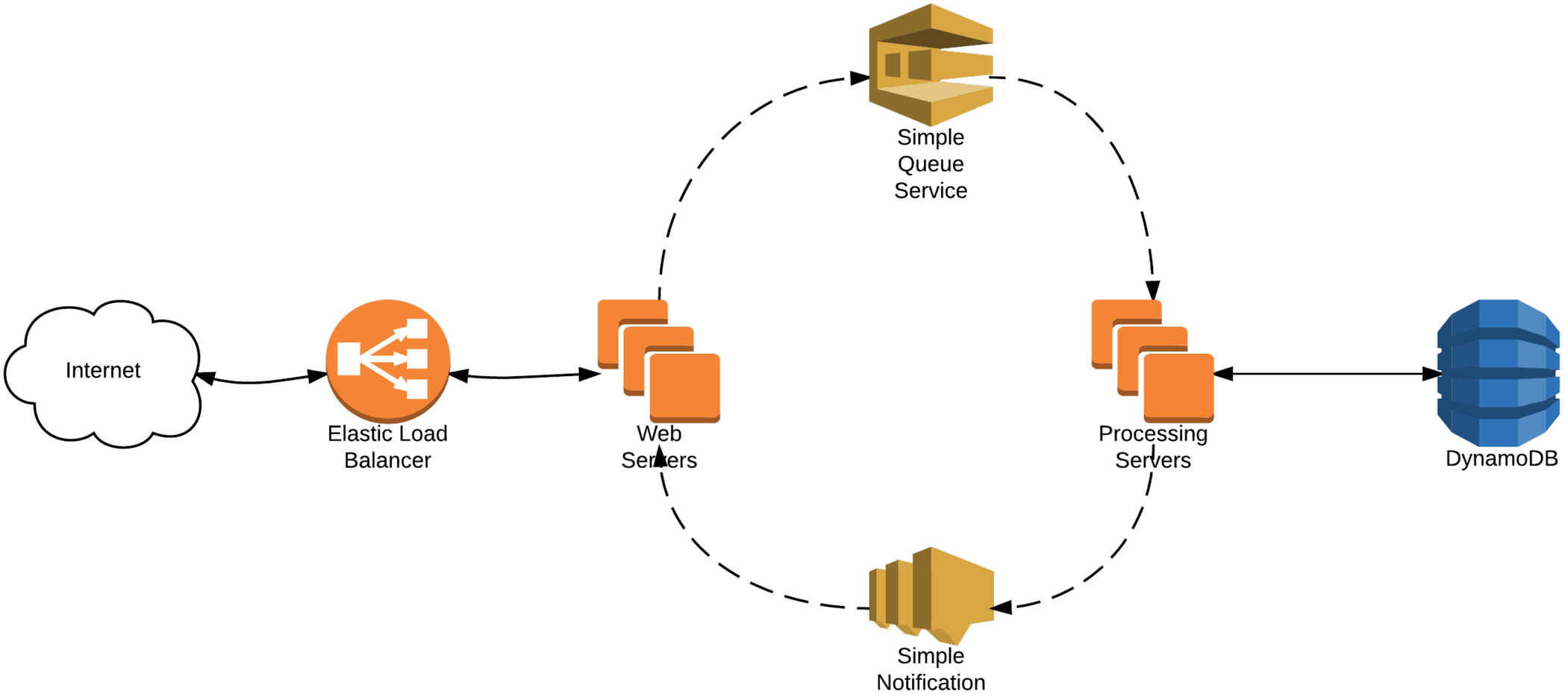
After an overly-flat network this is the most common mistake I see in assessments

Turn on native security controls/tools



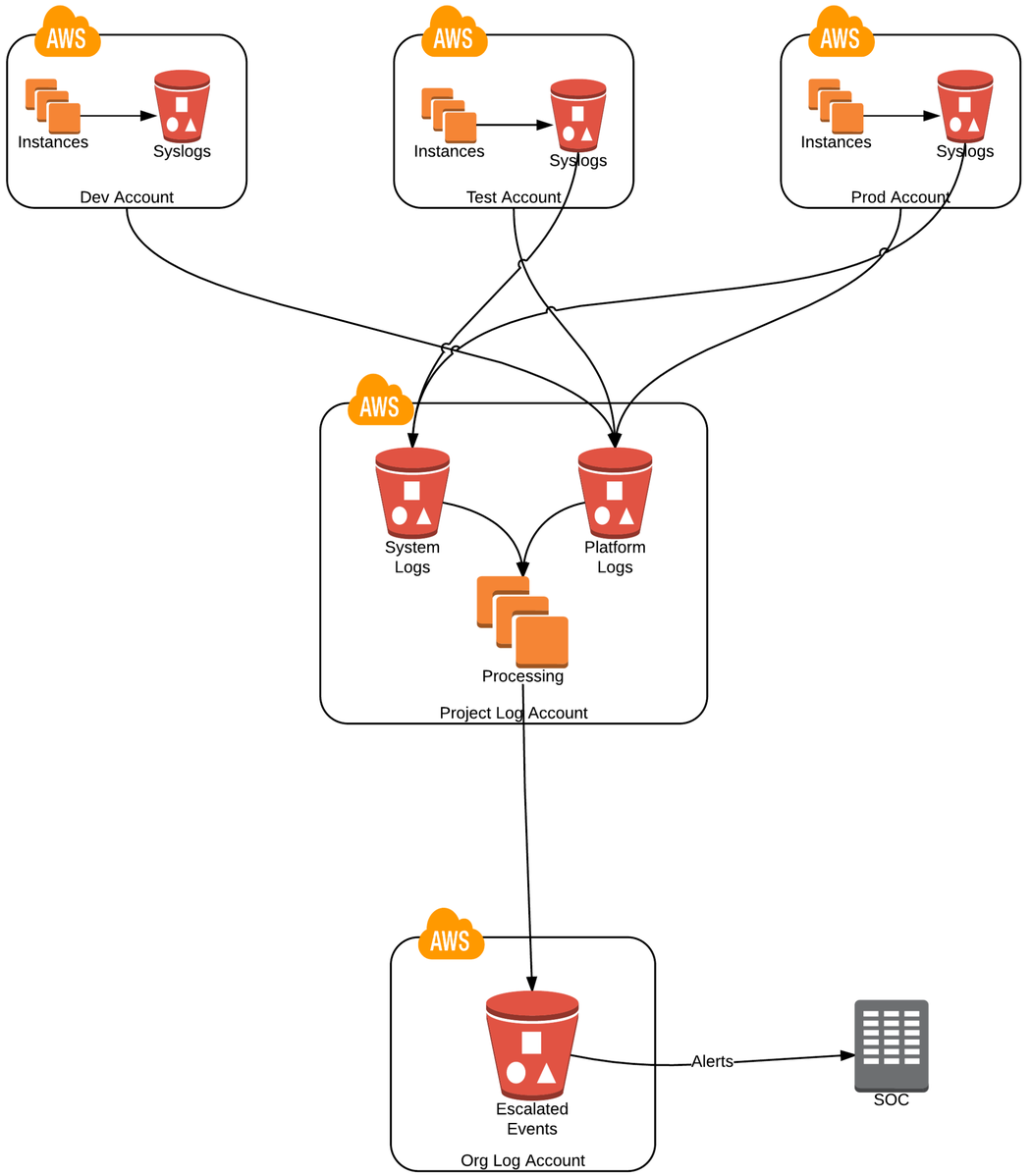
- Admin/API logs (CloudTrail, Audit Logs, Stackdriver)
- Threat intel/security center (e.g. GuardDuty, Azure Security Center)
- Assessment (Trusted Advisor, Azure Security Center, Cloud Security Command Center)
- Vulnerability assessment (Inspector)
- Other logs (service dependent, e.g. VPC Flow Logs, Stackdriver)
- DDoS/WAF
- Encryption
- Access controls
- Event-driven alerting
- …etc…

PaaS/Refactor



Build Shared Security Services

- Logging/Monitoring/Alerting
- Federated IAM
- Incident Response
- Vulnerability management
- NOT
  - Standard netsec (can use building blocks but should be specific to app needs)
  - Standard host security controls
    - See: Immutable

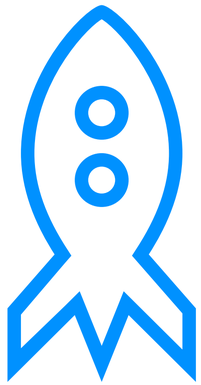


Set Guardrails

Guardrails

Continuously assess and enforce operational and security policies

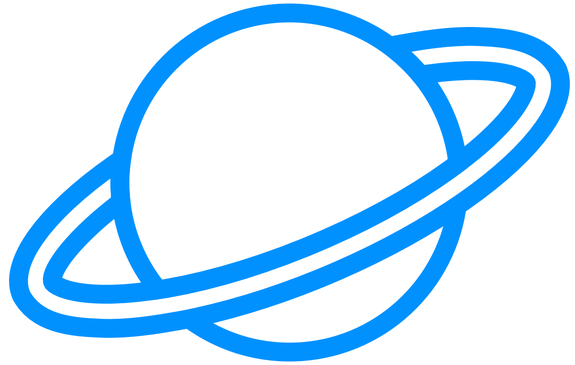
Fix security group or S3 misconfigurations



Workflows

Streamline and accelerate IT operations and security through automated workflows

Incident response



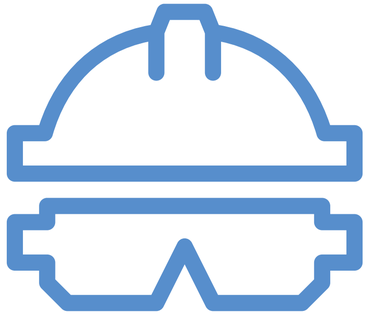
Orchestrations

Empower new capabilities through advanced orchestration of infrastructure, operations, and security

Automatic WAF insertion and configuration

Guardrails

- Define and set limits
  - Can be “allow” or “deny”
- Find deviations
  - Assessment or event based
- Evaluate the issue
- Fix/remediate
  - Automatically or manually depending on rules



Examples

- If you find a public S3 bucket, restrict it to our known network addresses
  - Unless it is approved or tagged
- Don’t allow internal security groups with all ports and protocols open in Prod
  - But allow in Dev
- Require MFA for API access for any user that needs MFA for console access
- Create our baseline IAM policies and roles for all new accounts
  - Based on the environment
- Validate that monitoring and alerting is properly configured
  - And fix if not
- Disable access keys that haven’t been used in 90 days
- Find instances with an IAM role that allows power user or greater access via API
  - Restrict the privileges
- Identify all cross-network peering from accounts we don’t own
  - Then check the security group permissions

Codify with infrastructure as code

vpclab18.template

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  KeyName:
    Description: The EC2 Key Pair to allow SSH access to the instance
    Type: String
Resources:
  LabVPC:
    Type: 'AWS::EC2::VPC'
    Properties:
      CidrBlock: 10.0.0.0/16
      InstanceTenancy: default
      EnableDnsSupport: 'true'
      EnableDnsHostnames: 'true'
      Tags:
        - Key: Name
          Value: Admin VPC
        - Key: Group
          Value: Vpclab
  subnet667a7420:
    Type: 'AWS::EC2::Subnet'
    Properties:
      CidrBlock: 10.0.0.0/24
      AvailabilityZone: us-west-2a
      MapPublicIpOnLaunch: 'true'
      VpcId: !Ref LabVPC
      Tags:
        - Key: Name
          Value: Admin public
        - Key: Group
```

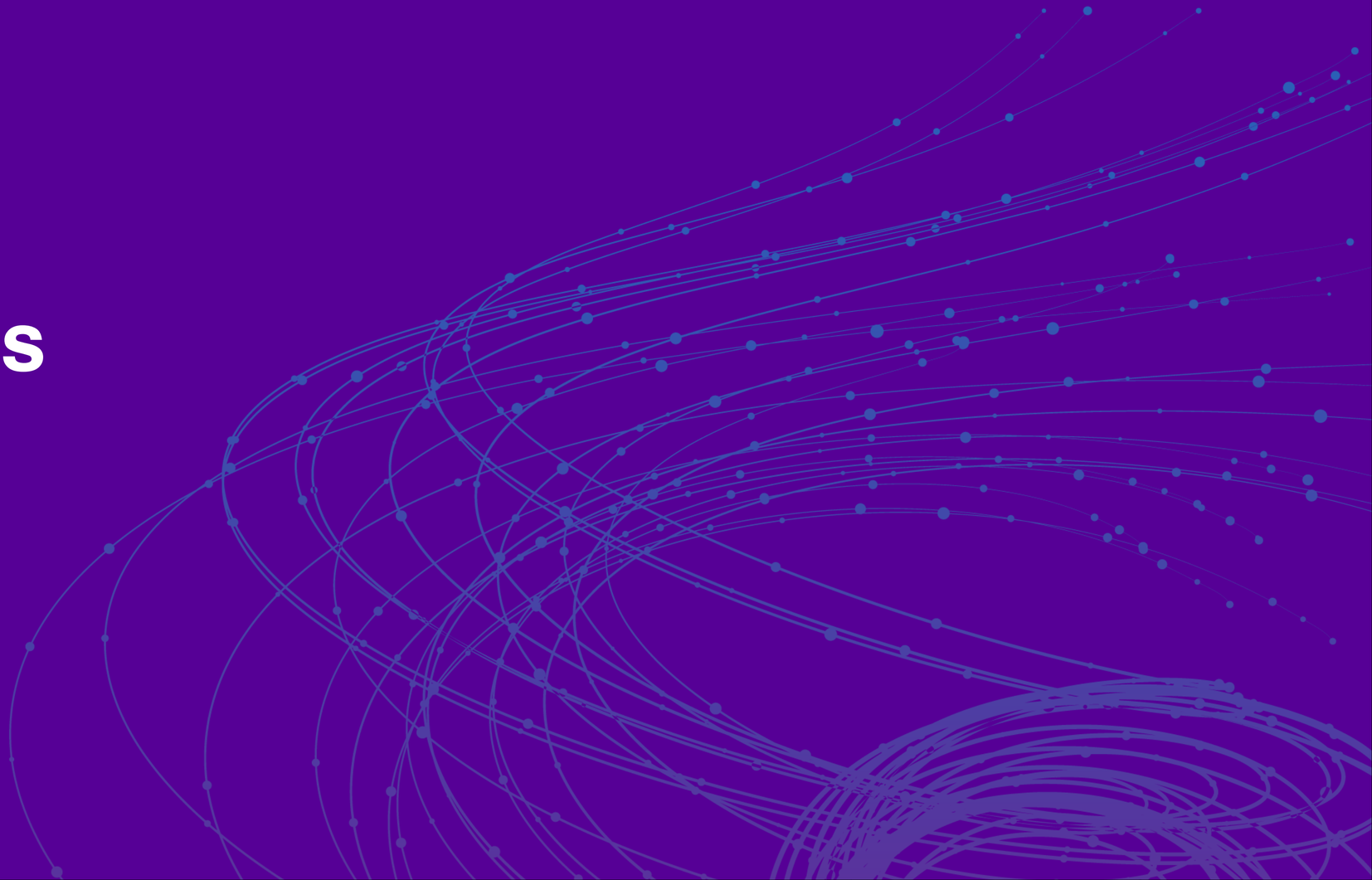
Line 105, Column 28 Spaces: 2 Plain Text

Determine expiration date

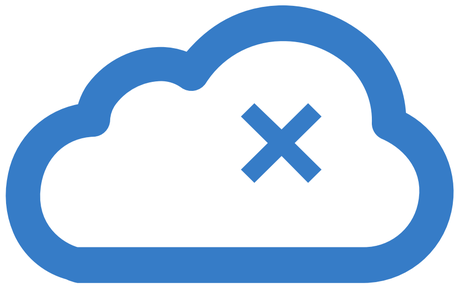
- Most lift and shifted projects should not be considered permanent
- Refactoring is an ongoing process
- Rebuilding in parallel is often an excellent long term option

RSAConference2019

Antipatterns



Top design errors

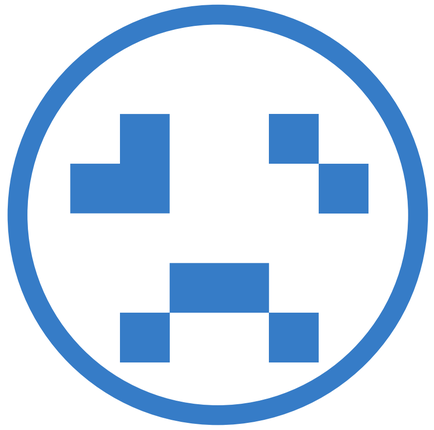


- Single account
- Forcing the application into a standard or crowded network
- Using traditional security tools instead of native due to lack of trust or understanding
- Not planning for service limits
- Requiring hybrid and defaulting to transit networks. Every time.

Top implementation errors

- Flat networks
- Overly-open IAM
- Any-any rules
- Improper native tool configurations
- Treating security groups like firewalls

Top security program errors



- No cloud-specific IR
- No cloud-specific alerting (time delays)
- No guardrails
- Trying to fix after implementation
- Not monitoring the management plane
- Ineffective or untested DR

Apply

- Next week you should:
  - Identify a self-constrained lift and shift project
    - Or an existing project to update
- In the first three months following this presentation you should:
  - Build your selection and strategy matrix for which pattern to use
  - Identify at-risk projects that used antipatterns (e.g. “a big cloud account with a lot of different projects inside)
- Within six months you should:
  - Begin using isolated environments for all new migrations
  - Build out shared services and operational plans to support new migrations
  - Develop a strategy for reviewing and repairing previous at-risk projects

RSAConference2019
San Francisco | March 4–8 | Moscone Center

SESSION ID: STR-T08

Lift and Shift, Don’t Lift and Pray: Pragmatic Cloud Migration Strategies

Rich Mogull

Analyst/CEO Securosis
VP of Product DisruptOps
rmogull@securosis.com