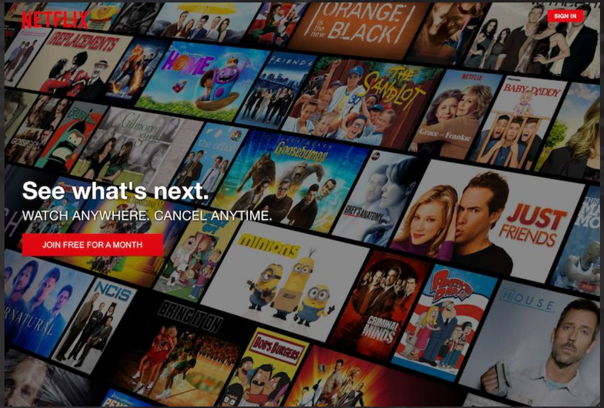
Scheduling a Fuller House: Container Management

Sharma Podila, Andrew Spyker - Senior Software Engineers

NETFLIX

About Netflix

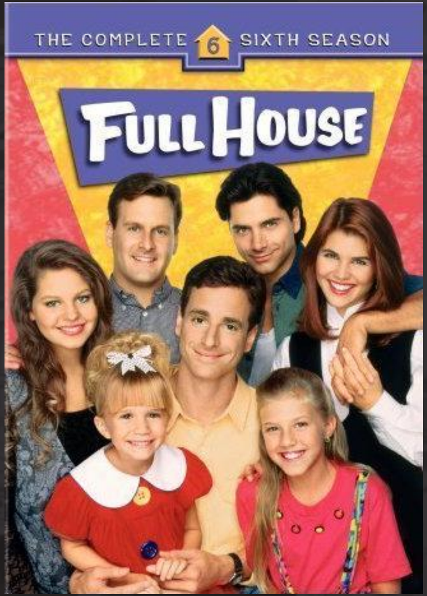
- 81.5M members
- 2000+ employees (1400 tech)
- 190+ countries
- > 100M hours watch per day
- > ⅓ NA internet download traffic
- 500+ Microservices
- Many 10’s of thousands VM’s
- 3 regions across the world



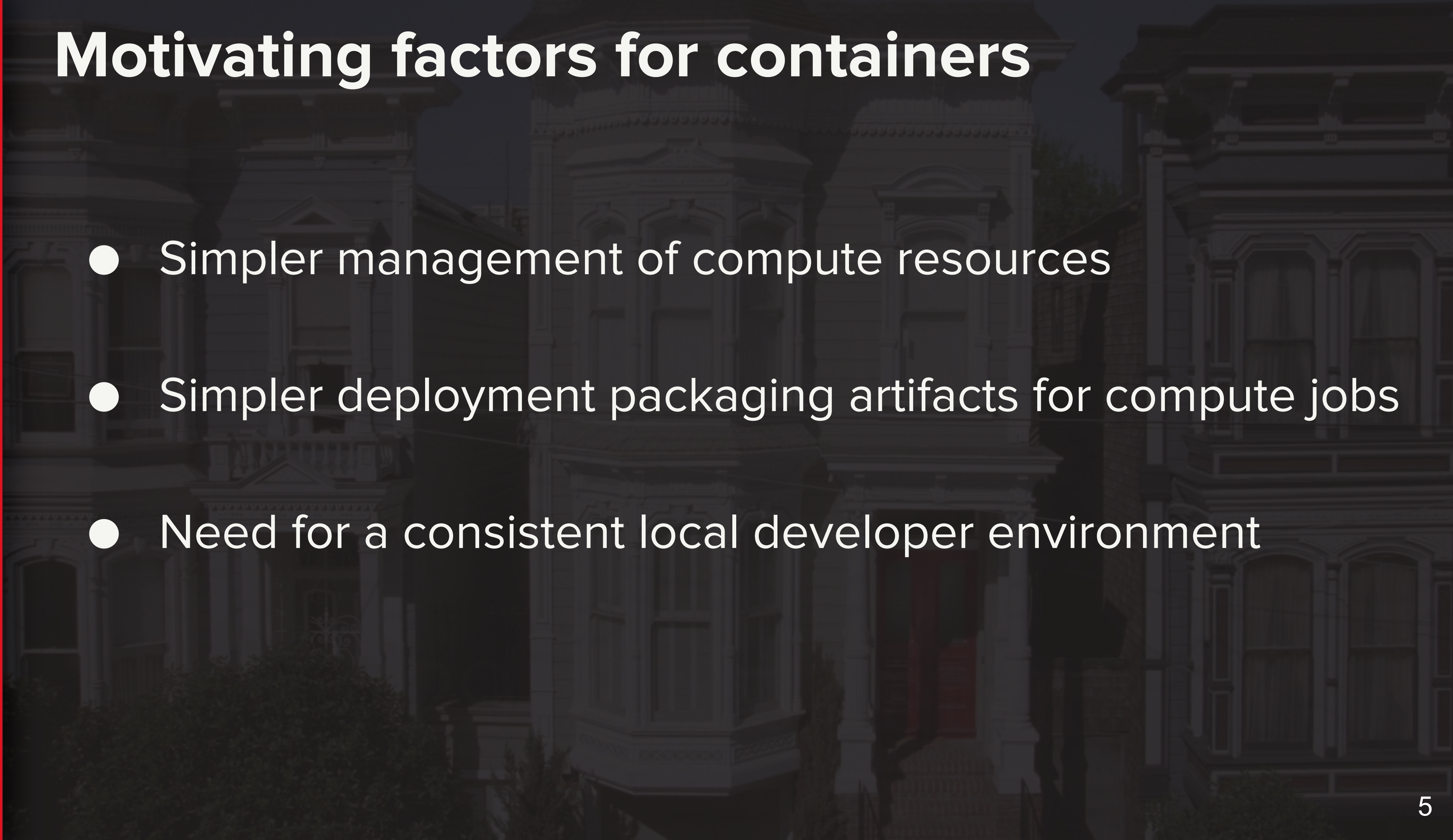
Agenda

- ⇨ Why containers at Netflix?
- What did we build and what did we learn?
- What are our current and future workloads?

Why a 2nd edition of virtualization?



- Given our resilient cloud native, CI/CD devops enabled, elastically scalable virtual machine based architecture, did we really need containers?



Motivating factors for containers

- Simpler management of compute resources
- Simpler deployment packaging artifacts for compute jobs
- Need for a consistent local developer environment

Simpler compute, Management & Packaging

Service style jobs (VM's)

- Use tested/secure base AMI
- Bake an AMI
- Define launch config
- Choose t-shirt sized instance
- Canary & red/black ASG's

Batch/stream processing jobs

- Here are the files to run my process
- I need m cores, n disk, and o memory
- Please just run it for me!

Consistent developer experience

- Many years focused on
  - Build, bake / cloud deploy / operational experience
  - Not as much time focused on developer experience

- New Netflix local developer experience based on Docker

- Has had a benefit in both directions
  - Cloud like local development environment
  - Easier operational debugging of cloud workloads

What about resource optimization?

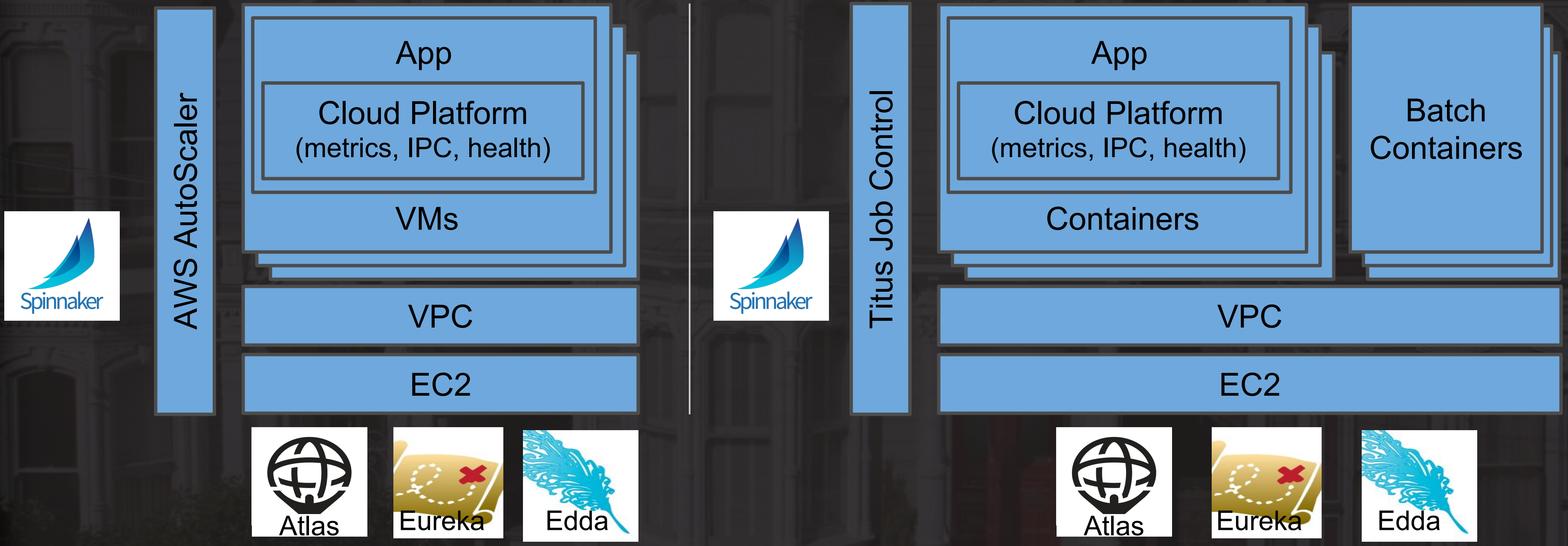
- Not absolutely required and easier to get wins at larger scale across larger virtual machine fleet

- However, *potential* benefits to
  - Elastic resource pool for scaling batch & adhoc jobs
  - Reliable smaller instance sizes for NodeJS
  - Cross Netflix resource optimizations
    - Trough usage, instance type migration

Agenda

- Why containers at Netflix?
- ⇨ What did we build and what did we learn?
- What are our current and future workloads?

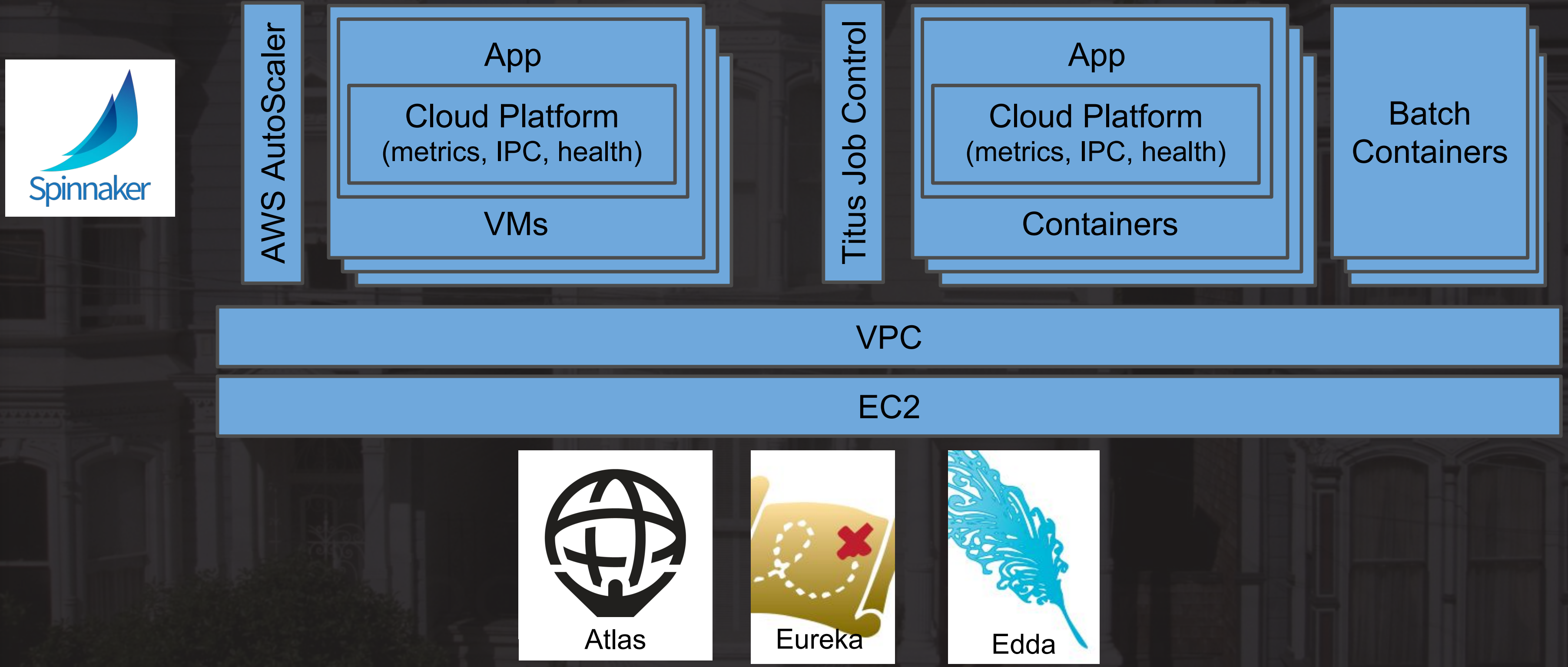
Lesson: Support containers by leveraging existing Netflix IaaS focused cloud platform



Existing - VM's

Titus - Containers

Why - Single consistent cloud platform

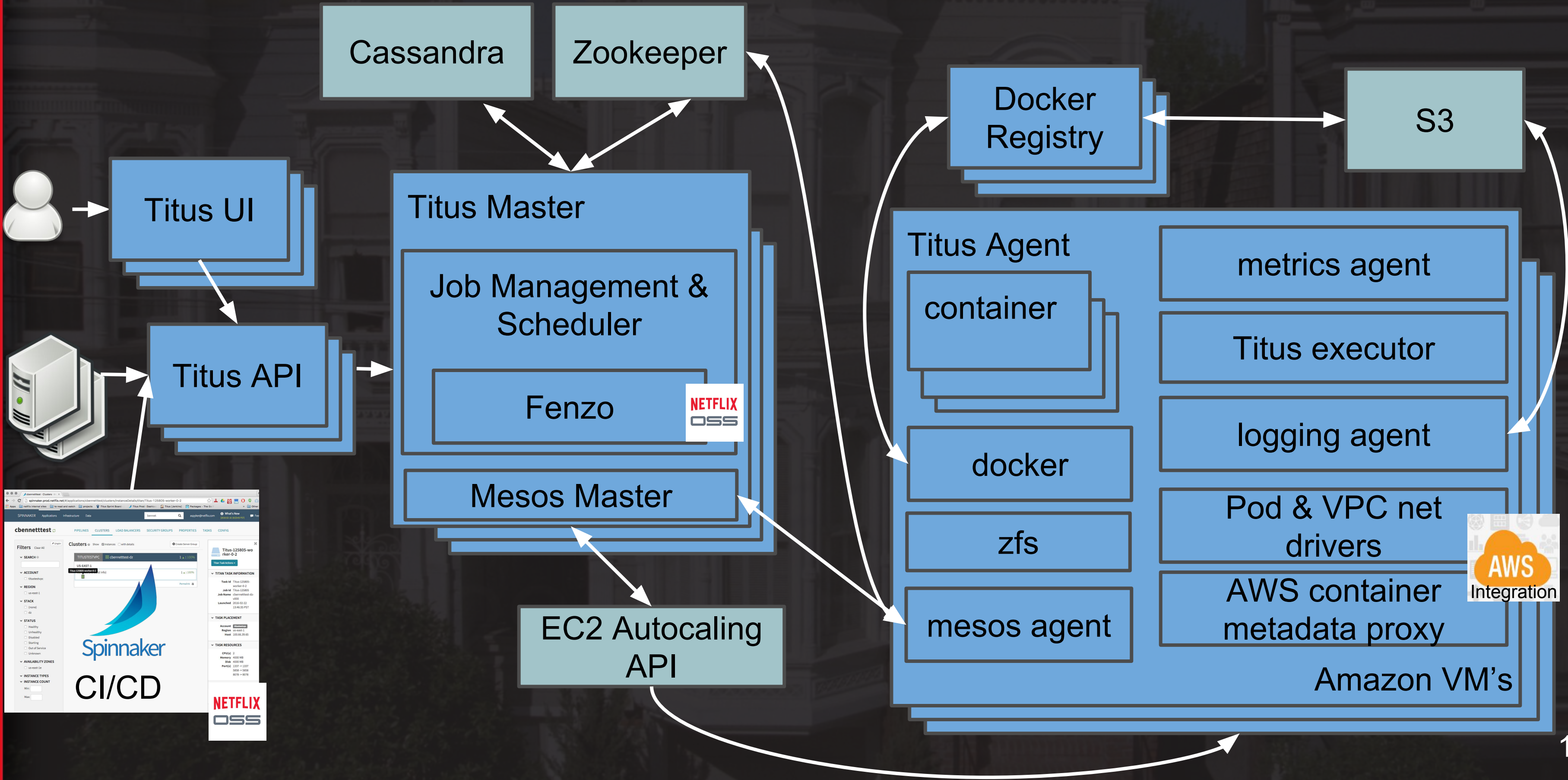


Netflix Cloud Infrastructure (VM’s + Containers)

Lesson: Buy vs. Build, Why build our own?

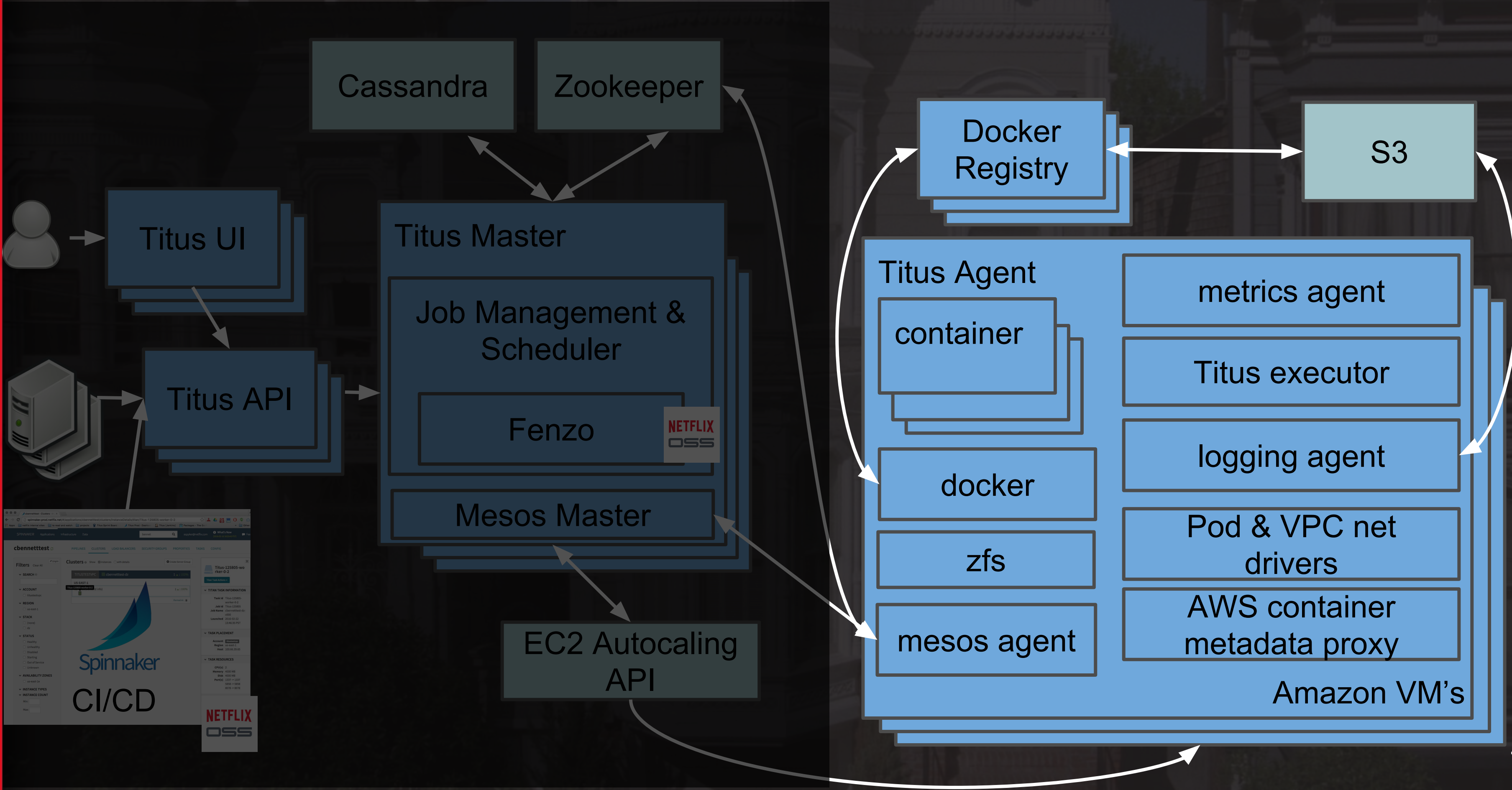
- Looking across other container management solutions
  - Mesos, Kubernetes, and Swarm
- Proven solutions are focused on the datacenter
- Newer solutions are
  - Working to abstract datacenter and cloud
  - Delivering more than cluster manager
    - PaaS, Service discovery, IPC
    - Continuous deployment
    - Metrics
  - Not yet at our level of scale
- Not appropriate for Netflix

"Project Titus" (Firehose peek)

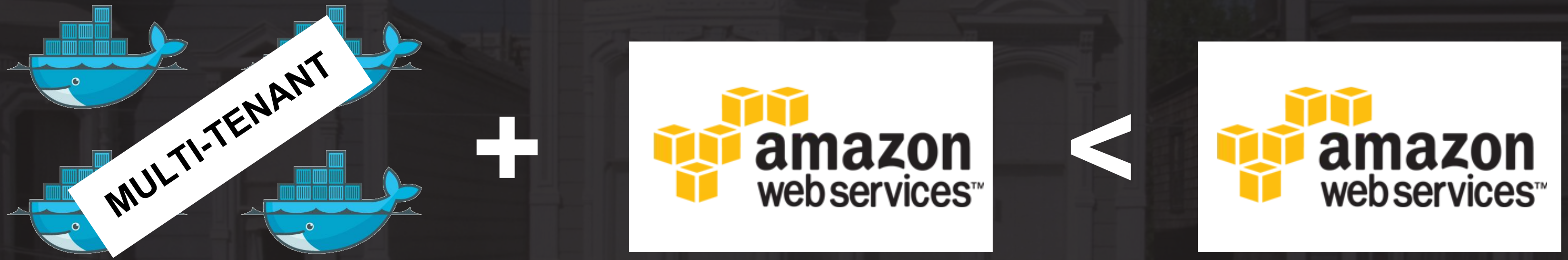


Is that all?

Container Execution



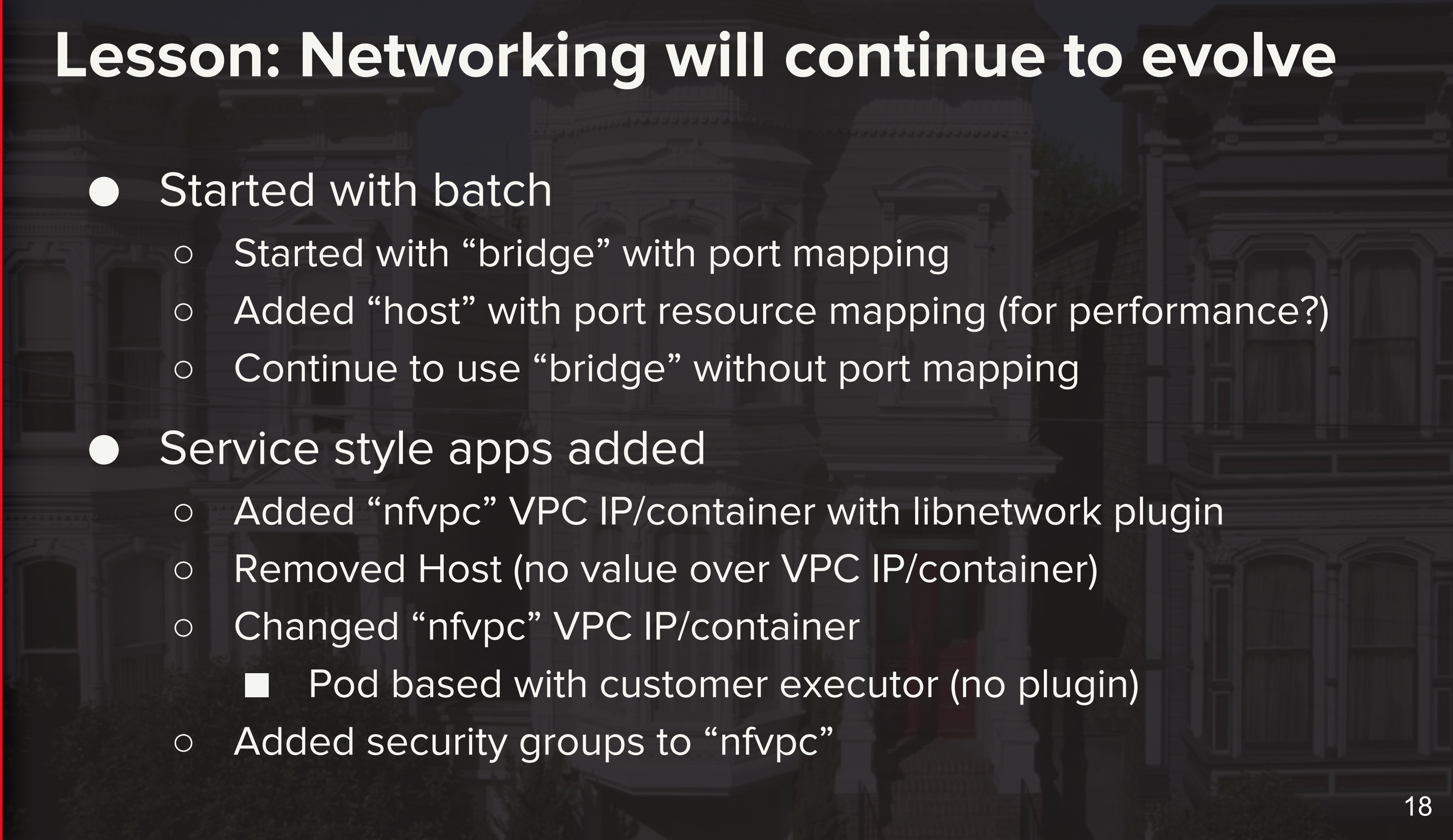
Lesson: What you lose with Docker on EC2



- Networking: VPC
- Security: Security Groups, IAM Roles
- Context: Instance Metadata, User Data / Env Context
- Operational Visibility: Metrics, Health checking
- Resource Isolation: Networking, Local Storage

Lesson: Making Containers Act Like VM's

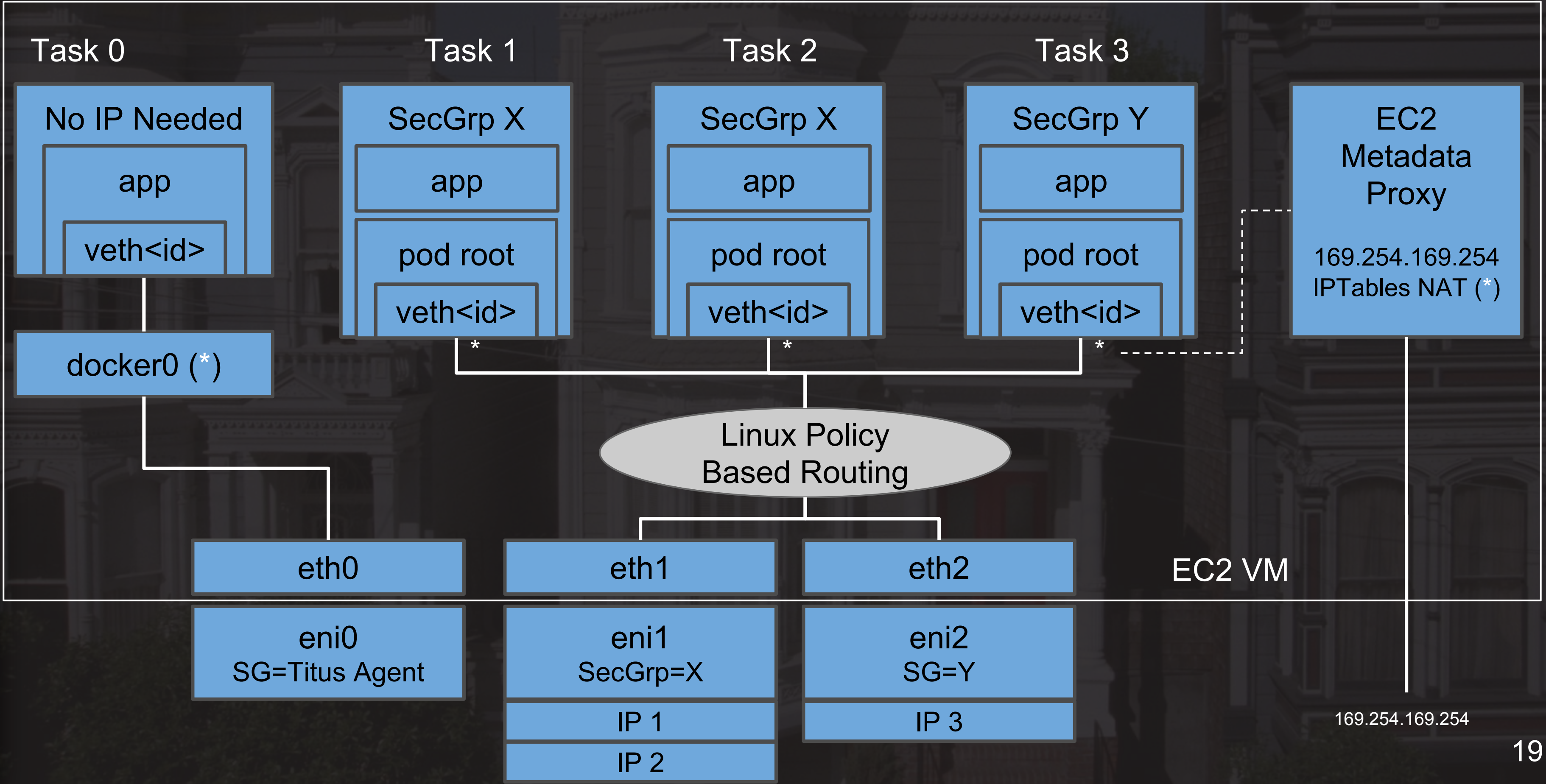
- Built: EC2 Metadata Proxy
  - Provide overridden scheduled IAM role, instance id
  - Proxy other values
- Provided: Provide Environmental Context
  - Titus specific job and task info
  - ASG app, stack, sequence, other EC2 standard
- Why? Now:
  - Service discovery registration works
  - Amazon service SDK based applications work



Lesson: Networking will continue to evolve

- Started with batch
  - Started with “bridge” with port mapping
  - Added “host” with port resource mapping (for performance?)
  - Continue to use “bridge” without port mapping
- Service style apps added
  - Added “nfvpc” VPC IP/container with libnetwork plugin
  - Removed Host (no value over VPC IP/container)
  - Changed “nfvpc” VPC IP/container
    - Pod based with customer executor (no plugin)
  - Added security groups to “nfvpc”

Plumbing VPC Networking into Docker



Lesson: Secure Multi-tenancy is Hard

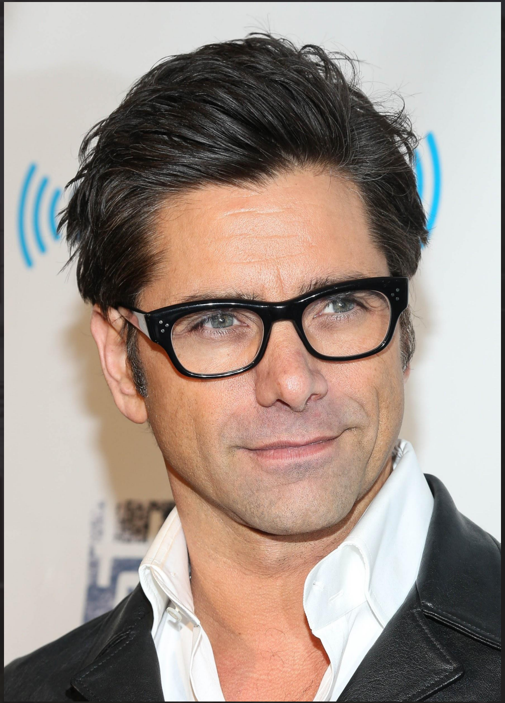
Common to VM's and tiered security needed

- Protect the **reduced** host IAM role, Allow containers to have specific IAM roles
- Needed to support **same** security groups in container networking as VM's

User namespacing

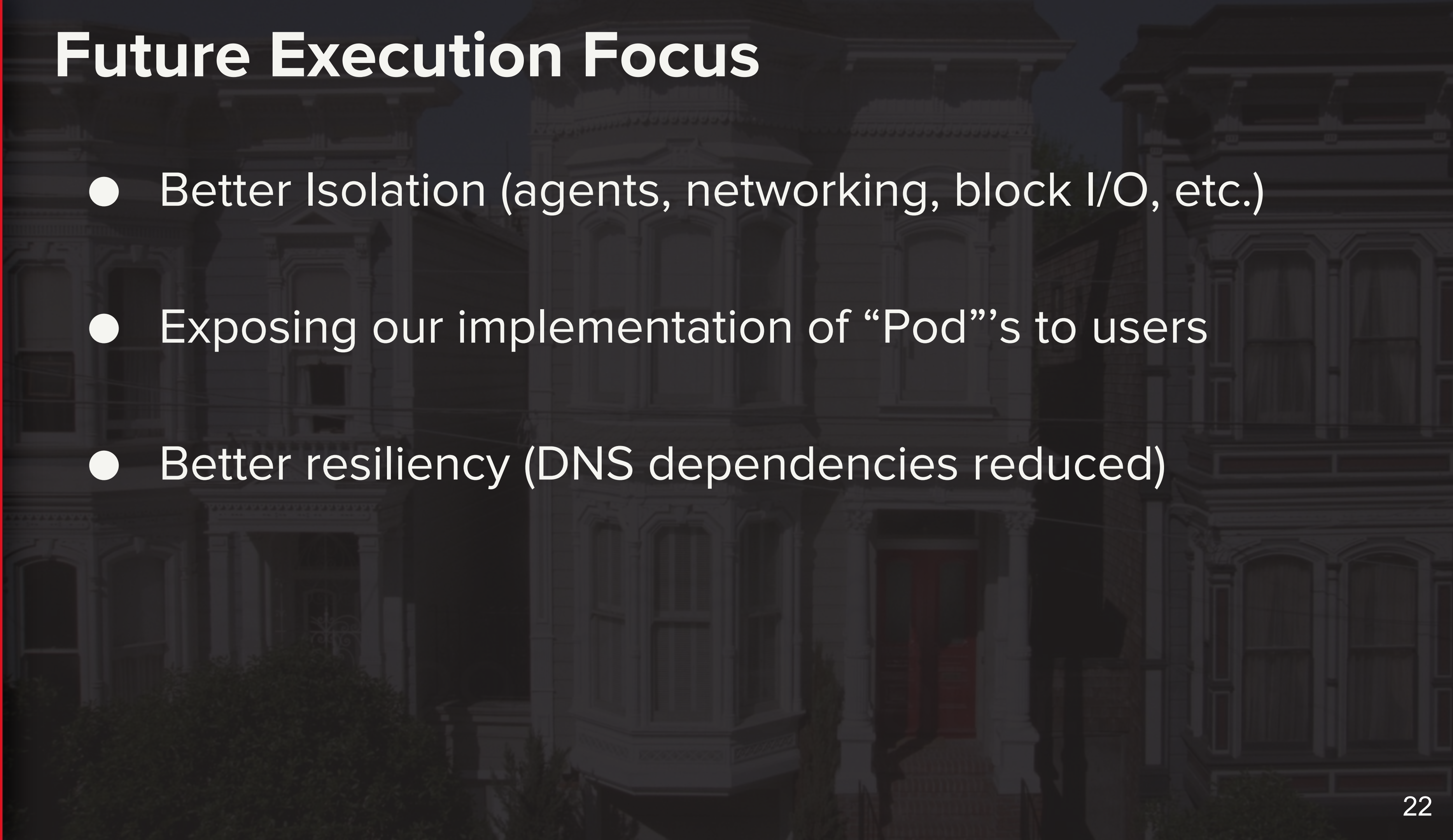
- Docker 1.10 - Introduced User Namespaces
  - Didn't work /w shared networking NS
- Docker 1.11 - Fixed shared networking NS's
  - But, namespacing is per daemon
  - Not per container, as hoped
- Waiting on Linux
  - Considering mass chmod / ZFS clones

Operational Visibility Evolution



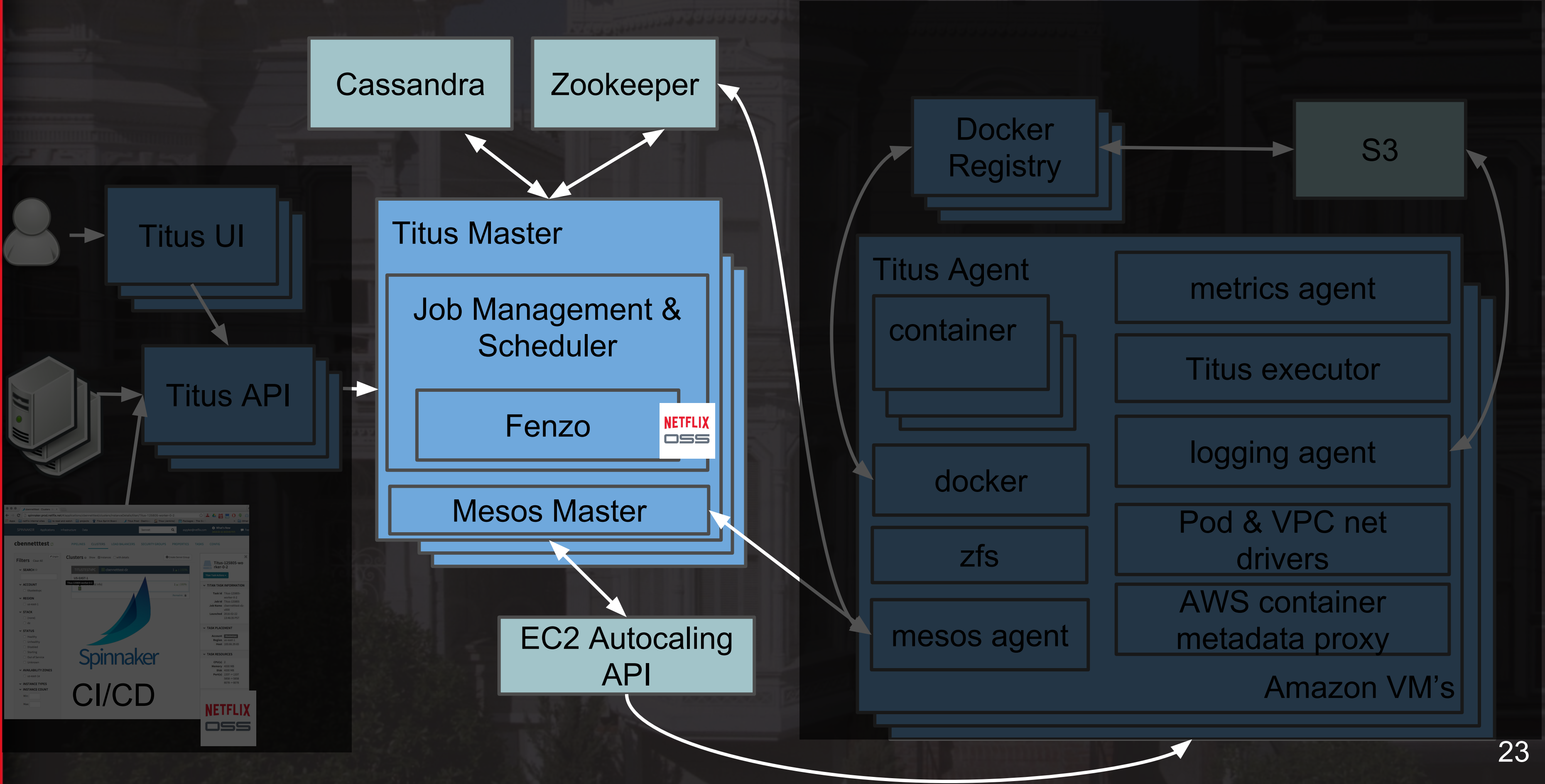
- What is “node” - containers on VM’s
- Soft limits / bursting a good thing?
    - Until percent util and outliers are considered
- System level metrics
    - Currently - hand coded cgroup scraping
    - Considering Intel Snap replacement
- Pollers - Metrics, Health, Discovery
    - Created Edda common “server group” view

Future Execution Focus



- Better Isolation (agents, networking, block I/O, etc.)
- Exposing our implementation of “Pod”’s to users
- Better resiliency (DNS dependencies reduced)

Job Management and Resource Scheduling



Lesson: Complexity in scheduling

- Resilience
  - Balance instances across EC2 zones, instances within a zone
- Security
  - Two level resource for ENIs
- Placement optimization
  - Resource affinity
  - Task locality
  - Bin packing (Auto Scaling)

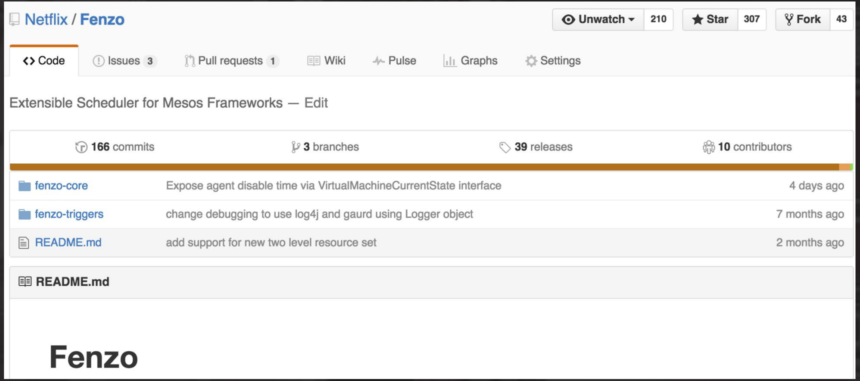
Lesson: Keep resource scheduling extensible

Fenzo - Extensible Scheduling Library

Features:

- Heterogeneous resources & tasks
- Autoscaling of mesos cluster
  - Multiple instance types
- Plugins based scheduling objectives
  - Bin packing, etc.
- Plugins based constraints evaluator
  - Resource affinity, task locality, etc.
- Scheduling actions visibility

https://github.com/Netflix/Fenzo


Netflix / Fenzo

Unwatch 210 | Star 307 | Fork 43

Code | Issues 3 | Pull requests 1 | Wiki | Pulse | Graphs | Settings

Extensible Scheduler for Mesos Frameworks — Edit

166 commits | 3 branches | 39 releases | 10 contributors

fenzo-core	Expose agent disable time via VirtualMachineCurrentState interface	4 days ago
fenzo-triggers	change debugging to use log4j and gaurd using Logger object	7 months ago
README.md	add support for new two level resource set	2 months ago

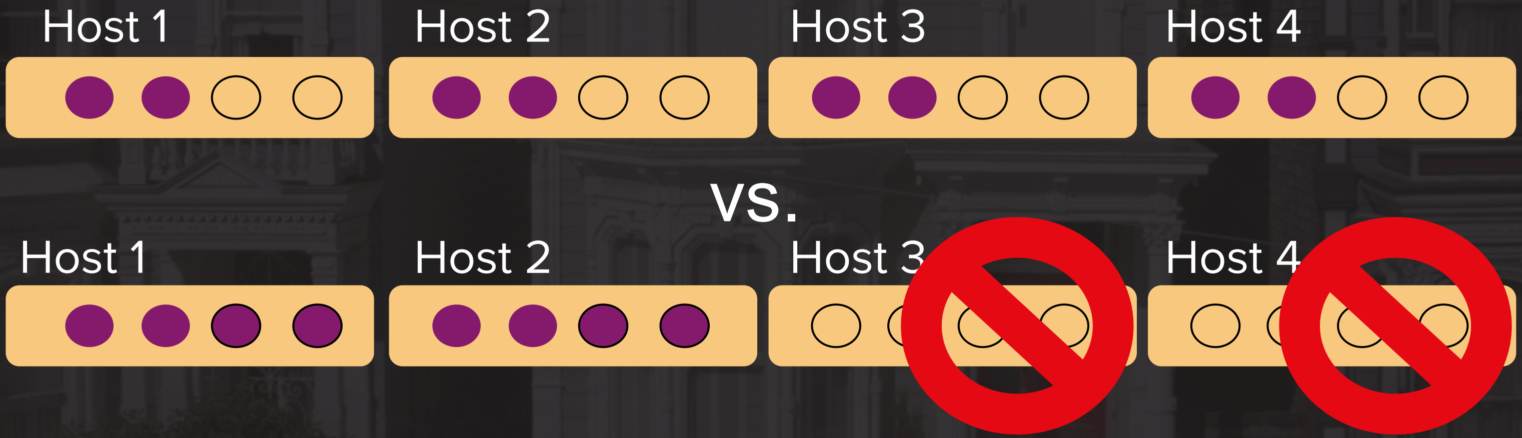
README.md

Fenzo

NETFLIX OSS

Cluster Autoscaling Challenge

For long running stateful services



Resources assigned in Titus

- CPU, memory, disk capacity
- Per container AWS EC2 Security groups, IP, and network bandwidth via custom driver
- Abstracting out EC2 instance types

Security groups and their resources

A two level resource per EC2 Instance: ***N*** ENIs, each with ***M*** IPs

ENI 0

Assigned Security Group: SG1 | Used IPs Count: 2 of 7

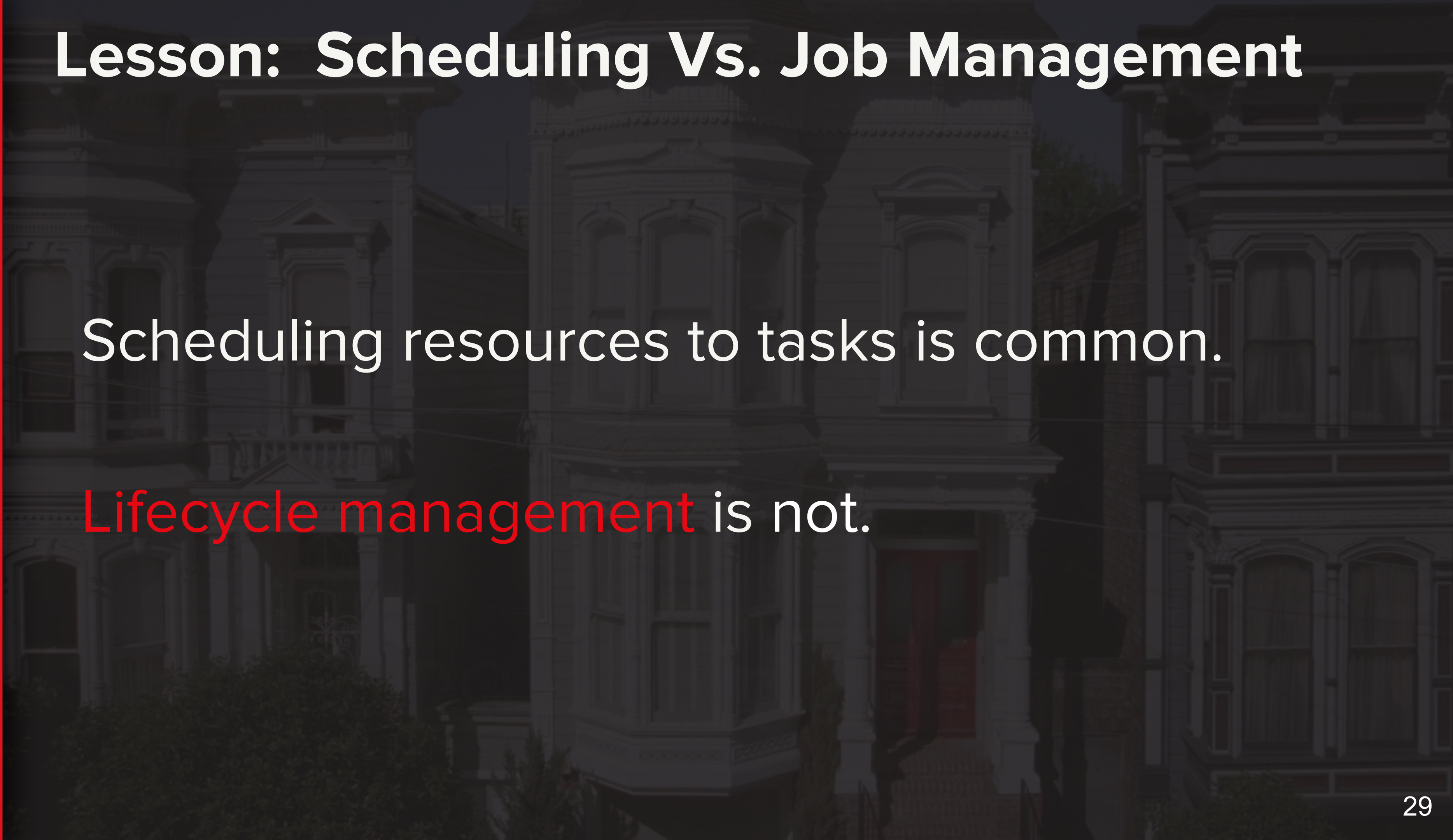
ENI 1

Assigned Security Group: SG1,SG2 | Used IPs Count: 1 of 7

ENI 2

Assigned Security Group: SG3 | Used IPs Count: 7 of 7

Lesson: Scheduling Vs. Job Management



Scheduling resources to tasks is common.

Lifecycle management is not.

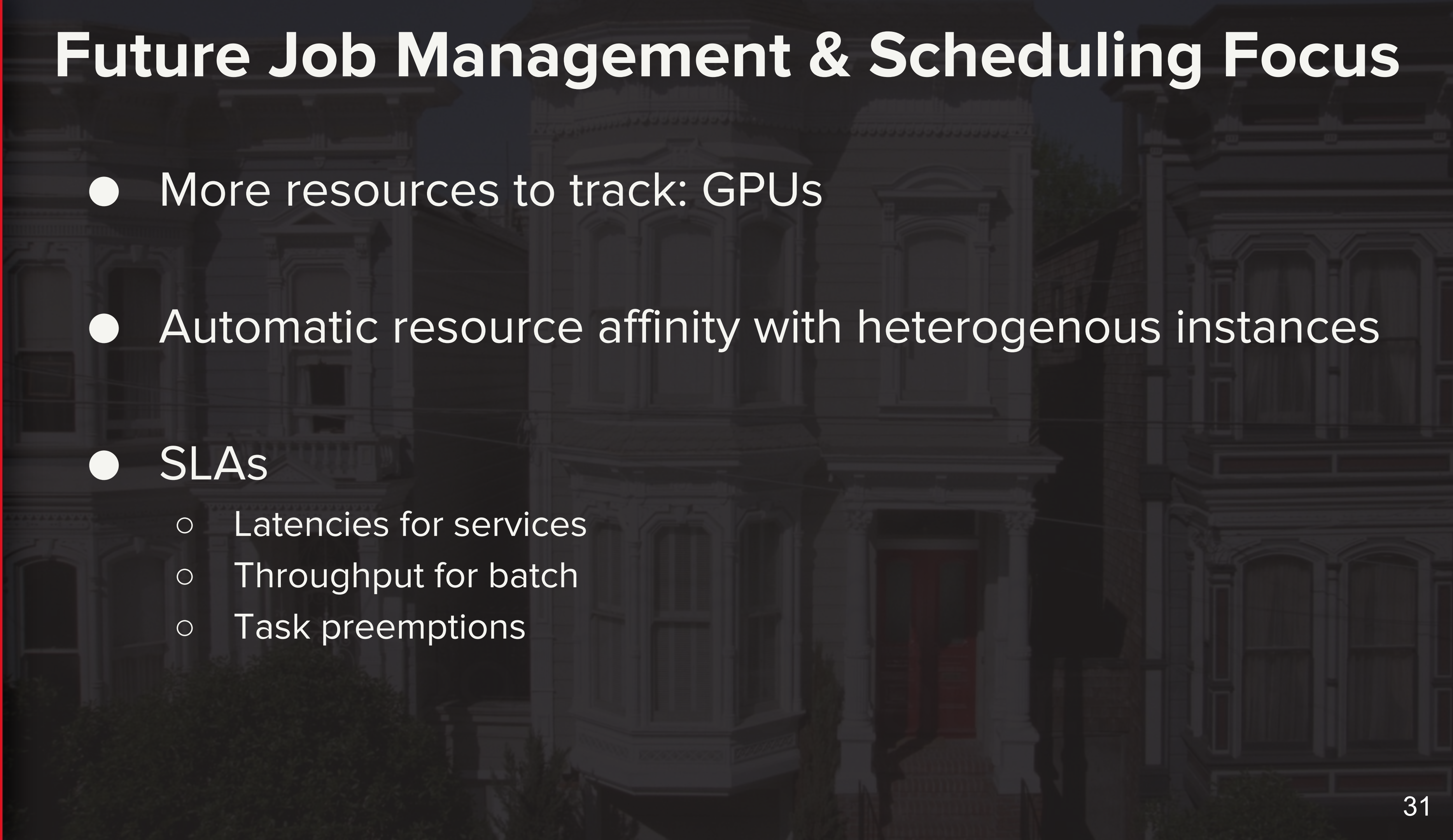
Lesson: Scheduling Vs. Job Management

Task scheduling concerns

- Assign resources to tasks
- Cluster wide optimizations
  - Bin packing
  - Global constraints, like SLAs
- Task preferences and constraints
  - Locality with other tasks
  - Resource affinity

Job manager concerns

- Managing task/instance counts
- Creating metadata, defining constraints
- Lifecycle management
  - Replace failed task executions
- Handle failures
  - Rate limit requeuing & relaunching
  - Time out tasks in transitionary states



Future Job Management & Scheduling Focus

- More resources to track: GPUs
- Automatic resource affinity with heterogenous instances
- SLAs
  - Latencies for services
  - Throughput for batch
  - Task preemptions

Things we didn't cover in this talk

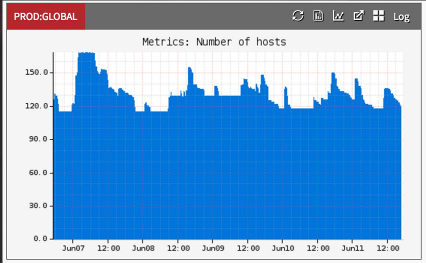
- Overall integration
  - Chaos, continuous delivery, performance insight
- Container Execution
  - Logging (live log access & S3 log rotation)
  - Liveness and health checking
  - Isolation (disk usage, networking, block I/O)
  - Image registry (metrics, security scanning)
- Scheduling
  - Autoscaling heterogeneous pools
  - Host-task fitness criteria
- API
  - Extensibility, polymorphic, SLA and job/container ownership

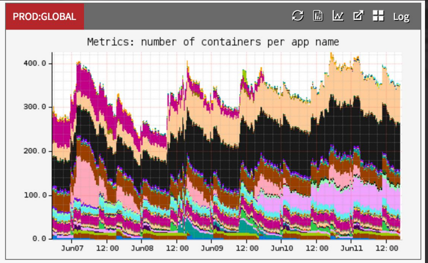
Agenda

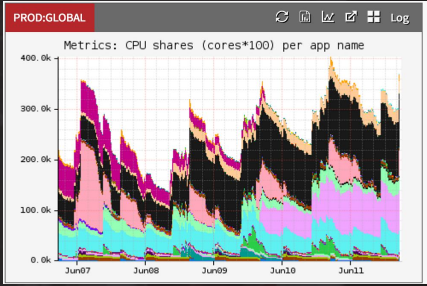
- Why containers at Netflix?
- What did we build and what did we learn?
- ➪ What are our current and future workloads?

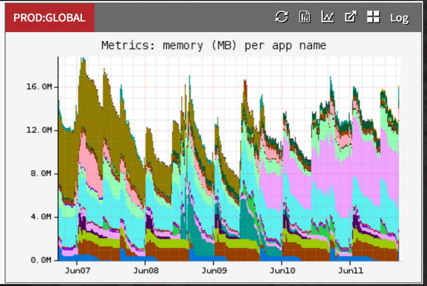
Current Titus Production Usage

- Autoscaling
  - 100’s of r3.8xl’s
  - Each 32 vCPU, 244G
- Peak
  - Thousands of cores
  - Tens of TB’s memory
- Thousands containers/day
  - ~ 100 different images





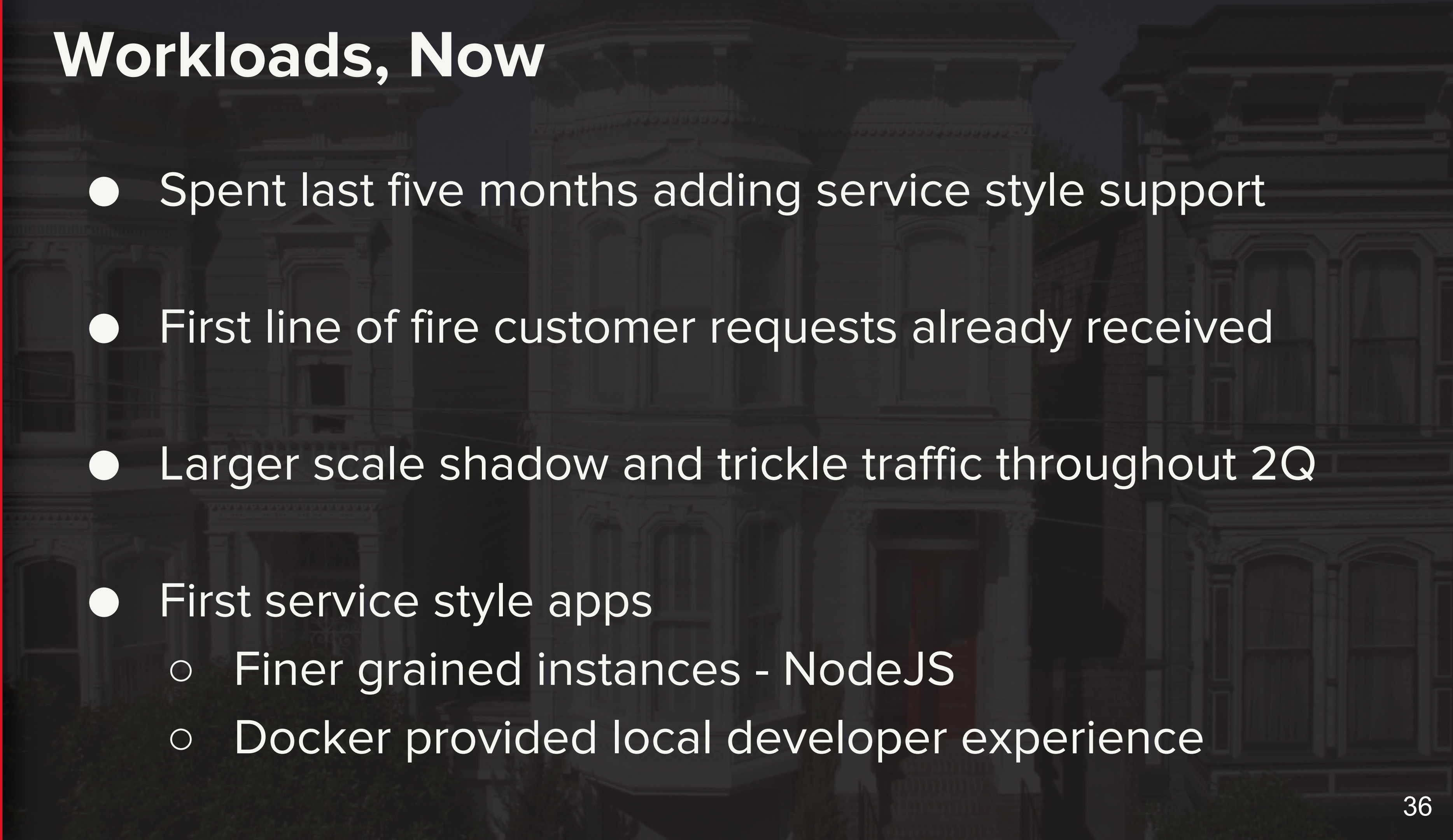




Workloads, Past

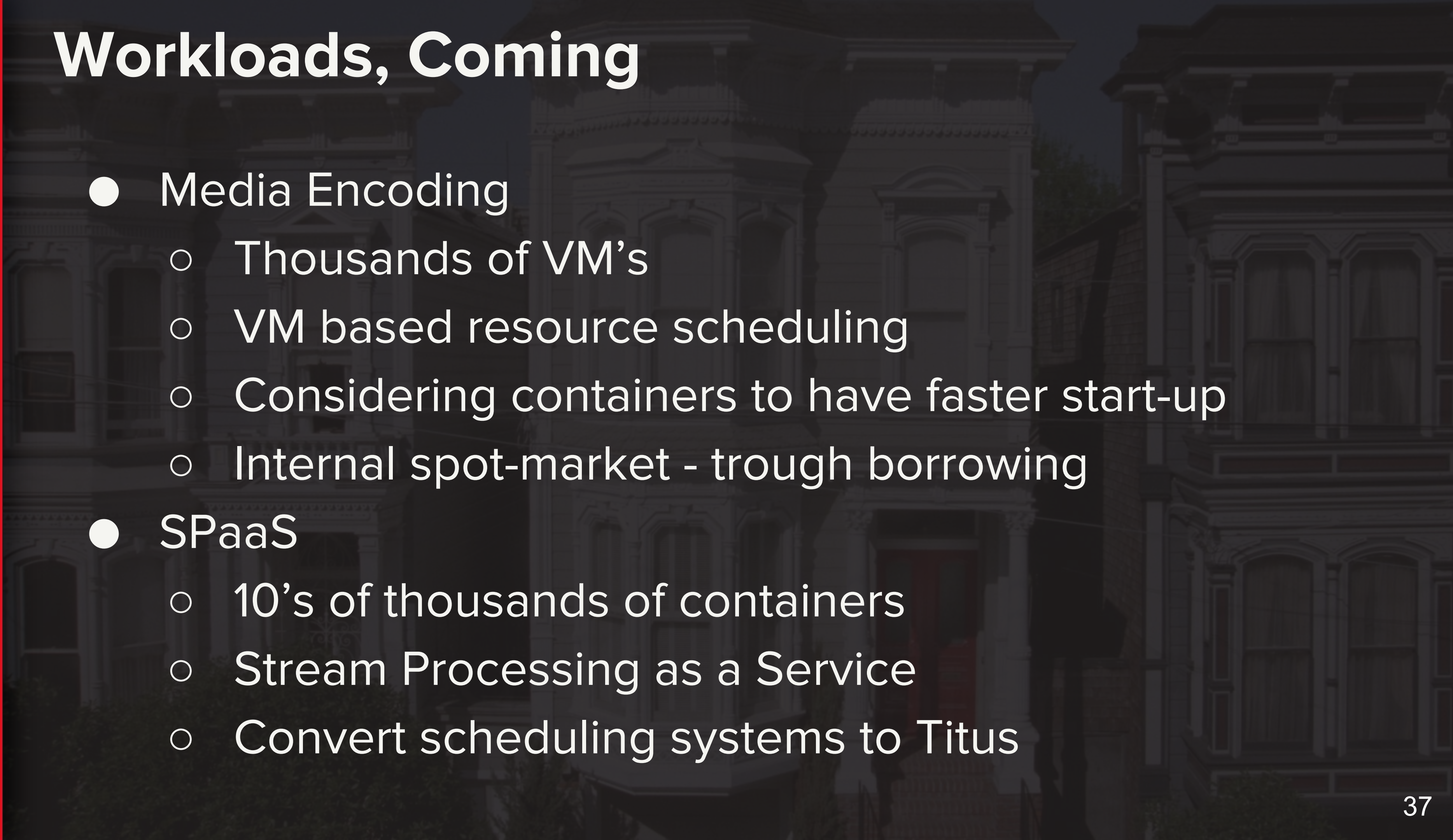
- Most current usage is batch
  - Algorithm training, adhoc reporting jobs

- Sampling:
  - Training of “sims” and A/B test models
  - Open Connect Device/IX reporting
  - Web security scanning and analysis
  - Social media analytics updates



Workloads, Now

- Spent last five months adding service style support
- First line of fire customer requests already received
- Larger scale shadow and trickle traffic throughout 2Q
- First service style apps
  - Finer grained instances - NodeJS
  - Docker provided local developer experience



Workloads, Coming

- Media Encoding
  - Thousands of VM’s
  - VM based resource scheduling
  - Considering containers to have faster start-up
  - Internal spot-market - trough borrowing
- SPaaS
  - 10’s of thousands of containers
  - Stream Processing as a Service
  - Convert scheduling systems to Titus

Questions?

Other Netflix QCon Talks

Title	Time	Speaker(s)
The Netflix API Platform for Server-Side Scripting	Monday 10:35	Katharina Probst
Scheduling A Fuller House: Container Mgmt @ Netflix	Tuesday 10:35	Andrew Spyker & Sharma Podila
Chaos Kong - Endowing Netflix with Antifragility	Tuesday 11:50	Luke Kosewski
The Evolution of the JavaScript	Wednesday 4:10	Jafar Husain
Async Programming in JS: The End of the Loop	Friday 9:00	Jafar Husain