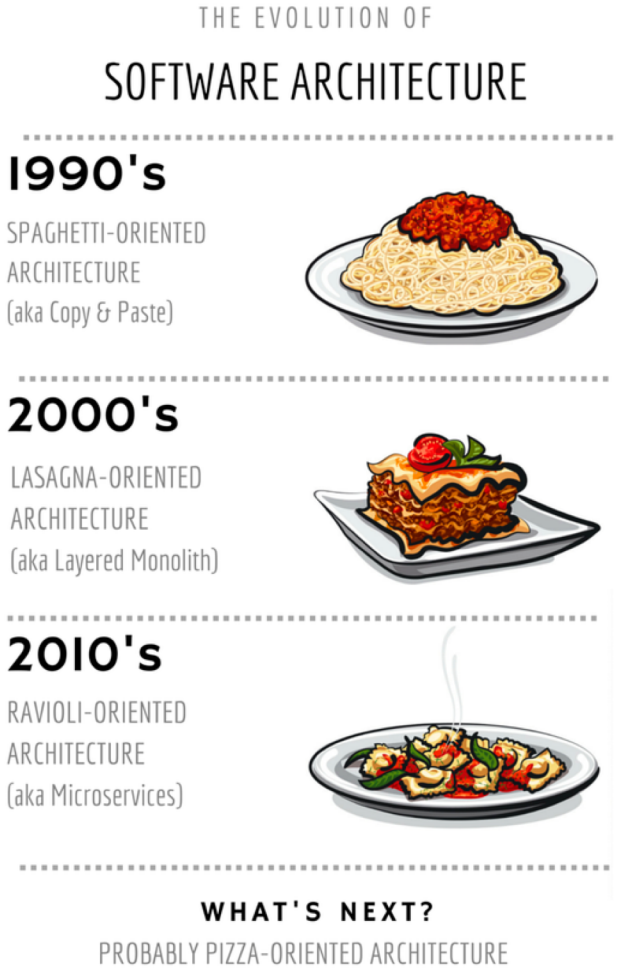
WELCOME TO DAY 3

OPERATING IN A CAAS

module 07

nebulaworks

Section 7.1 | app design considerations



> where is the app on the spectrum?

you need borders and tools<

the devops border

Eliminating the blame game of "it works on my machine."

- should be clear
- contain feedback mechanisms
- should be end-to-end

the tools team

- script ninja
- infrastructure plumber
- performance magician
- analytics & detective

<- Netflix SimianArmy logo

- https://github.com/Netflix/SimianArmy

Chaos Engineering

- failure is unavoidable
- it must be tested
- it must be handled gracefully

Section 7.2 | Docker networking

container models are different

>>>WARNING<<<<
containers are not VMs

you have a choice…

… of network models

Networking Fundamentals

- Container Network Model (CNM)
    - Open source
    - Docker implemented libnetwork (written in go) based on CNM
    - libnetwork (OSS) includes: service discovery, basic loading balancing for containers
- Container Network Interface (CNI)
    - Used by rkt

- Docker supports container-to-container networks, as well as connecting to existing networks as well as connecting to existing networks and VLANs

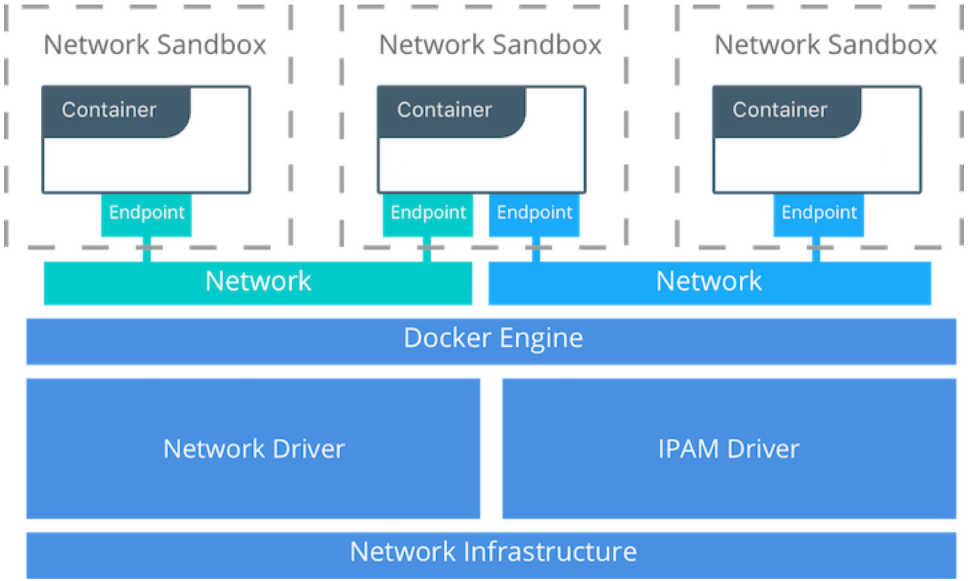
Container Network Interface

- Came from CoreOS and used as part of Rkt
- Simple interface between container runtime and network implementation
- This is now maintained by the Cloud Native Computing Foundation (CNCF)

Docker Networking Fundamentals

- CNM constructs
  - Sandbox
    - Contains the configuration of a containers network stack
    - Isolated network stack that includes
      - Routing tables, ports, interfaces, and DNS Config
  - Endpoint
    - An endpoint joins a sandbox to a Network
    - The connection to the network is abstracted
  - Network
    - Software implementation of a bridge/switch
    - Group together and isolate a collection of endpoints

Container Networking Model



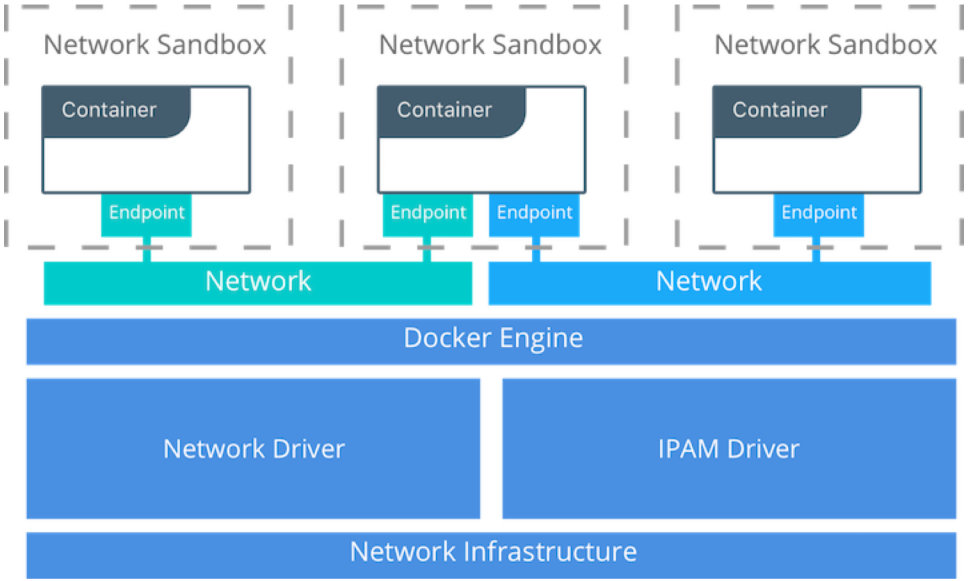
Credit: Mark Church thanks mark!

Docker Networking Design

CNM Driver Interfaces

- CNM provides two interfaces for the OSS community, and 3rd party vendors to add functionality
    - Remote Drivers
        - Vendor/3rd party plugins
    - Native Drivers
        - Responsible to make networks work, and can support various drivers
            - Out of the box drivers can be used e.g bridge, overlay
- IPAM (IP Address Management Driver)
    - This driver provides default IP addresses or subnets for endpoints and networks
    - Leverage docker network, container service commands to manually assign IP addresses

Container Networking Model



Credit: Mark Church thanks mark!

Docker Networking

- We can use networking plugins to extend functionality
- Docker supports some common cases
    - Single host bridge networks
    - Connecting to VLANs
    - Multi-host overlays

Docker Network Drivers

Linux

- Bridge (layer 2 switch)
- Overlay
- Macvlan
- Access to network plugins

Windows

- Nat
- Overlay
- Transparent
- l2bridge

docker network command

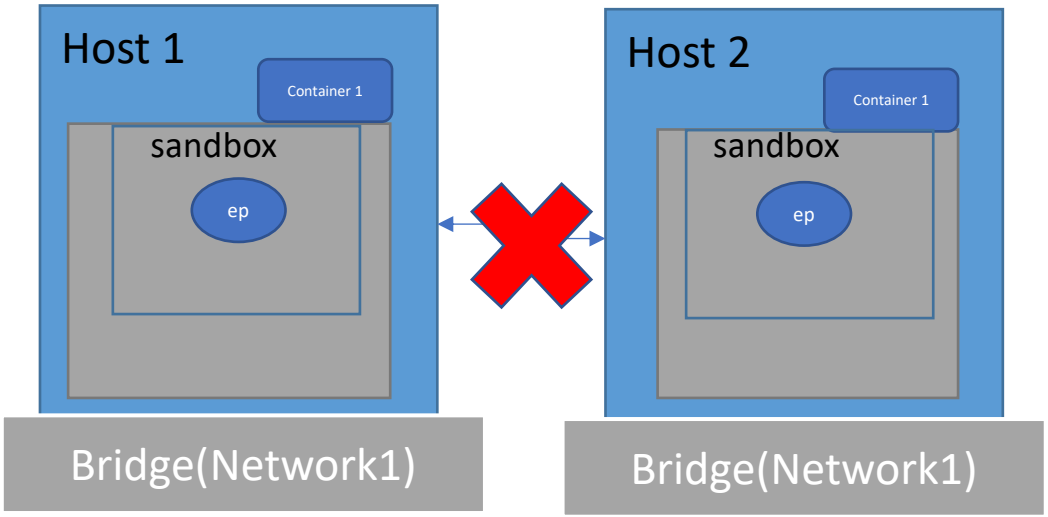
- `docker network connect`
- `docker network create`
- `docker network disconnect`
- `docker network inspect`
- `docker network ls`
- `docker network prune`
- `docker network rm`

Lab 4

nebulaworks

Docker Network Drivers

- Bridge
  - By default docker uses the **docker0** bridge
  - Can create custom bridge networks



Overlay Networks

- Bridge cannot create a distributed network to connect to multiple daemon hosts

- When a swarm is created..
  - Upon either an initialization or a join, two networks are created on the Docker host
    - **ingress** overlay network that manages data traffic related to swarm services (swarm mesh)
      - This is the default network used when a service is created
    - **docker_gwbridge** that connects a daemon to other daemons part of the swarm

Docker Swarm Mode

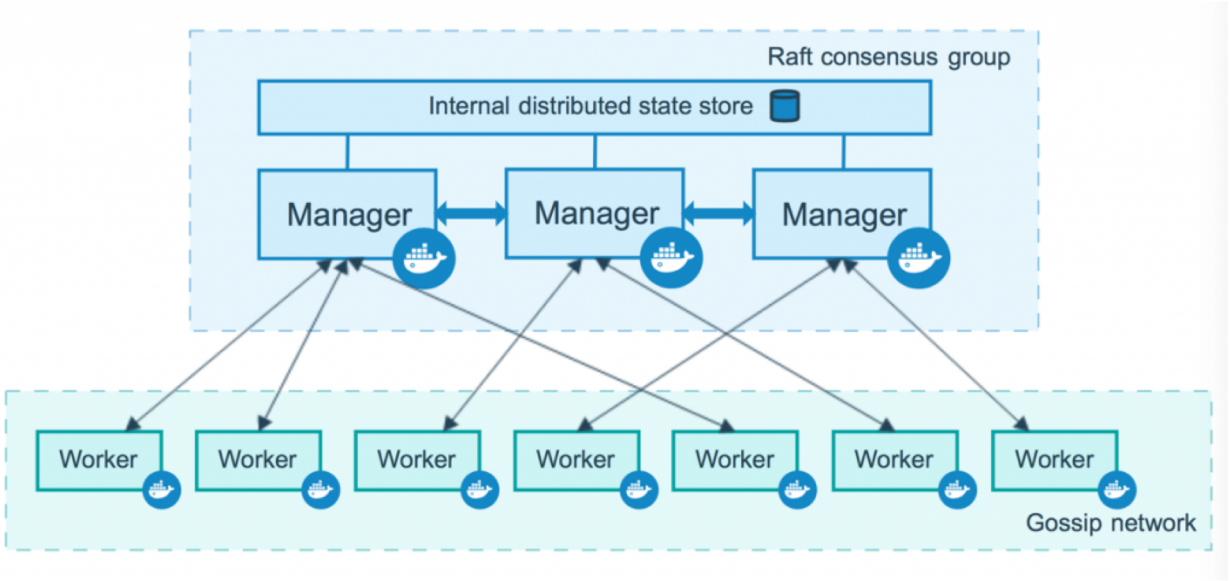
- Cluster management integrated with the Docker Engine
  $ docker swarm init
- Supports declarative **service** (more on this later**)** manifests
- Enables scaling
  - Define number of tasks you want to run
  - Feedback loops in manager monitors that desired state is maintained
- Service Discovery
  - Services are assigned DNS names and performs load balancing
- Each swarm node member enforces TLS authentication to encrypt communication between between other nodes and itself
- Uses an overlay network to for distributed nodes

Swarm Details

- Manager – Worker
- Docker needs a quorum (majority) for managers
  - The minimum number of hosts that a consensus group needs in order to be allowed to perform an operation
  - If you do not meet cluster quorum requirements workloads should not work
  - (N/2) +1
  - Only use odd numbers for managers

Managers	Quorum	FT
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2

Docker Swarm Mode



Source : Docker Documentation

Docker Swarm- Raft Protocol

- Consensus algorithm
- Leader (manager) election
- Log replication
- Low learning curve (vs something like Paxos)
- You should be able to easily learn the protocol, if you wanted to (more simple than Paxos)
- Etcd, the key value store is backed by Raft
- Raft is the algorithm that managS the global cluster state

Creating a Swarm

```
docker swarm init
```

- Switches the current node into swarm mode,
- Creates a swarm named default
- Configures the current node to be the leader manager in the swarm.
- Starts the internal distributed data store for Engine participating in the swarm.
- Generates self-signed root CA for internal swarm communication (default).
- Generates join-tokens for worker and manager nodes!

Joining nodes to a swarm

- Joining as a worker

```
$ docker swarm join --token <worker-token>
```

- Joining as a manager

```
$ docker swarm join --token <manager-token>
```

- Retrieving join-tokens on your lead manager node

```
$ docker swarm join-token worker (manager)
```

Locking Up a Swarm Cluster

- Despite swarms default out-of-the box security settings, a cluster can still be compromised if an old manager rejoins the cluster.
- Old managers joining a cluster will automatically decrypt and gain access to the raft log time-series db. They could then restore old backups and wipe the current swarm configuration (which would make the cluster unusable).
- In order to prevent scenarios like one described above, docker swarm gives you the option to lock your swarm cluster
- Locking a swarm cluster forces managers that been restarted to present a cluster-unlock key before they are allowed back into the cluster.

Locking Up a Swarm Cluster

- You can apply a lock to your swarm at any time.
- When it is first created

```
docker swarm init --autolock
```

- Any point after creation you can also update your swarm

```
docker swarm update --autlock=true
```

- If you don't want the autolock feature remove it by updating your cluster.

```
docker swarm update --autolock=false
```

Locking Up a Swarm Cluster

- It's important that you keep the cluster key somewhere safe. If you do lose your key, and you need to reconnect a manager to the cluster, you can view the key from an existing manager in your cluster. However, if the key has been rotated since your manager left the swarm, or your swarm cluster has fallen out of quorum then you're out of luck.

- Displaying the unlock-key (from a manager that is currently a part of the swarm cluster)

```
$ docker swarm unlock-key
```

- You also have the option to rotate your unlock keys (you should)

```
$ docker swarm unlock-key --rotate
```

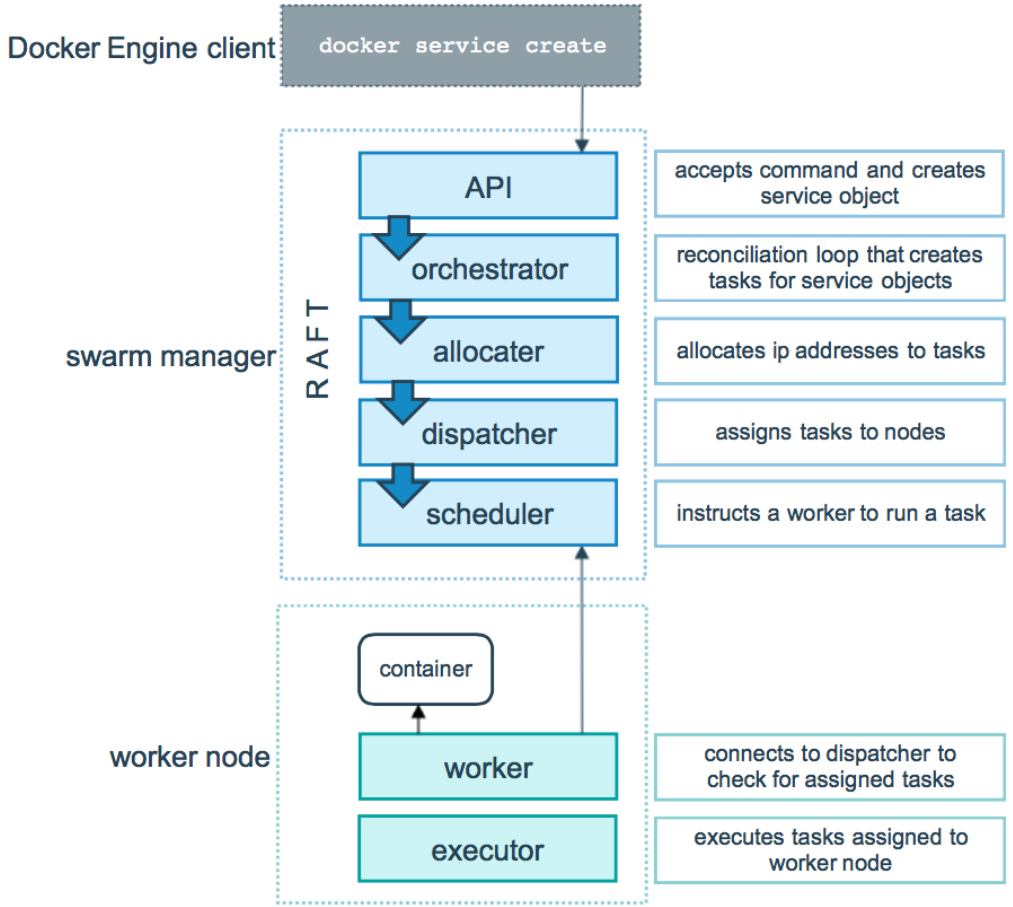
docker service (utilizing your swarm cluster)

- Goes hand-in-hand with Docker swarm
- It is a higher-level construct that wraps some advanced features around containers.
  - The ability to scale
  - Rolling updates
  - Roll backs
- Uses a declarative model
  - YOU define the desired state of your service and swarm will maintain this desired state.

Tasks

- The atomic unit of scheduling within a swarm.
- You can think of them as replicas or instances of your application.
- When you specify a desired state for your service (in your `docker service` create command), the orchestrator will realize this state by scheduling tasks.
- One-directional mechanism
  - Tasks progress through a series of states : assigned, prepared, running, etc.
  - If the tasks fails, the scheduler will remove it and start a new one!

How the swarm mode accepts `service create` requests and schedules tasks to worker nodes.



Types of Service Deployments

Replicated

- For a replicated service, you specify the number of tasks you want to run across your cluster. This is the default type of service deployment.

Global

- A global service deployment will run one task on every node in the cluster.

You can configure the service mode by using the `--mode` flag when creating a service

```
docker service create --name my-service --replicas 4 --mode
replicated (global) <image_name>
```

Scaling your Service

- There are two cli commands that will let you scale a service, with one having slightly more functionality than the other.

- The docker service update command allows you to scale one service at a time.

```
$ docker service update replicas=4 <service_name>
```

- The docker service scale command allows you to scale multiple services at the same time.

```
$ docker service scale <service1-ID>=<#-of-tasks> <service2-id>=<#-of-tasks>
```

Updating your Service

- There are various ways you can update your service. One of which, is applying a rolling update to your service. This can be used if you want your service to use a new-and-improved image, and you want to update it gracefully.
- You configure your rolling update policy at service deployment time.

```
$ docker service create \
    --replicas 10 \
    --name redis
    --updtate-delay 10s \
    --update-parallelism 2 \
    redis:3.05
```

- Our rolling update policy is modified by the update-policy-related flags we pass in!

Updating your Service

- The `update-delay` flag configures the time delay between updates to a service task or set of tasks.
- The `update-parallelism` flag configures the max number of service tasks that can be updated at the same time

- Now we can update our service to use the latest redis image.

```
$ docker service update --image redis:3.07 redis
```

Networking – Docker Swarm and ingress networks

- Docker lets you configure the **ingress** with docker version 17.05 and higher
- If the subnet conflicts with an existing network, you can recreate the **ingress** network

```
$ docker network rm ingress
```

WARNING! Before removing the routing-mesh network, make
sure all nodes in your swarm run the same docker engine version.
Otherwise, removal may not be effective and functionality of newly created
ingress networks will be impaired.

Are you sure want to continue? [y/N]

- Then create a new network
- Can only have 1 **ingress** network per host

Networking – Customize the docker_gwbridge

- This virtual bridge is what connects the ingress and overlay networks to a Docker daemon's physical network
- Docker_gwbrige is a kernel feature on the Docker host (it is not a Docker device)

Docker Swarm Mode

- Based on a project called SwarmKit
  https://github.com/docker/swarmkit/
- Introduced in Docker version 1.12

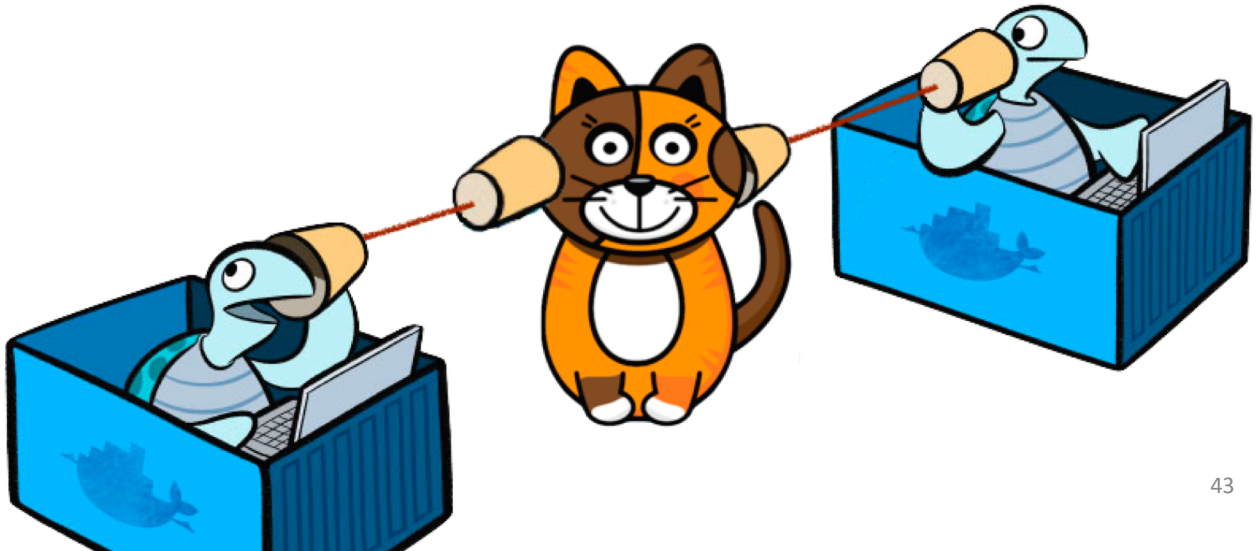
- https://github.com/docker/swarmkit/

Docker Networking Design

- Networking model based on CNM
- Provides portability different infrastructures

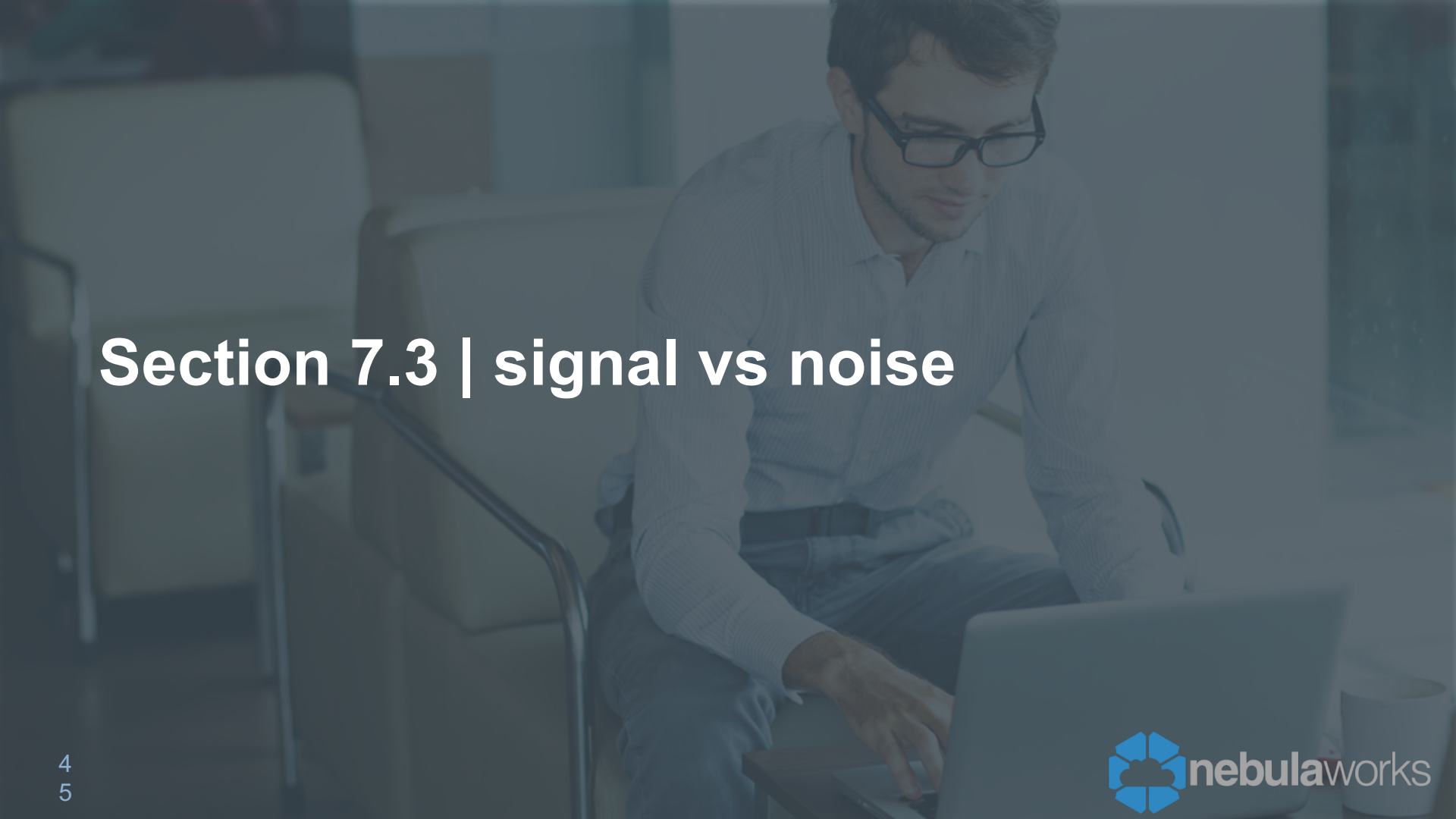
manual/direct connect

- macvlan
- ipvlan
- host network
- other

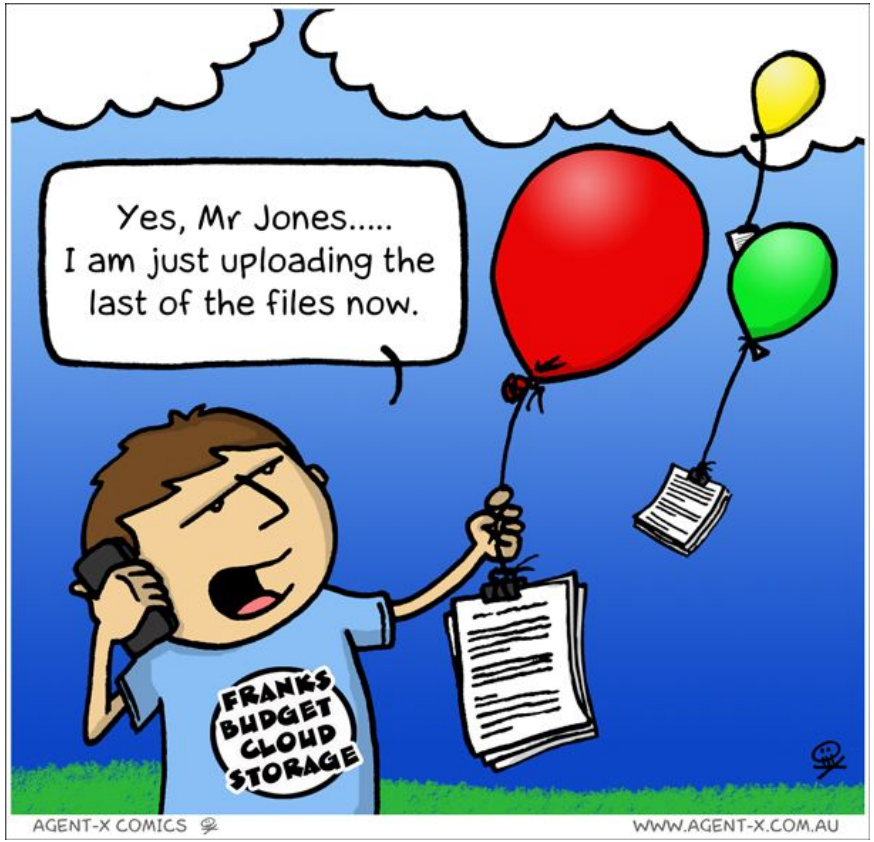


service network sensitivity

- management network
- orchestration
- overlay sdn
  - interconnects
  - load balancing
- infrastructure services
  - storage
  - databases
  - message queues



Section 7.3 | signal vs noise

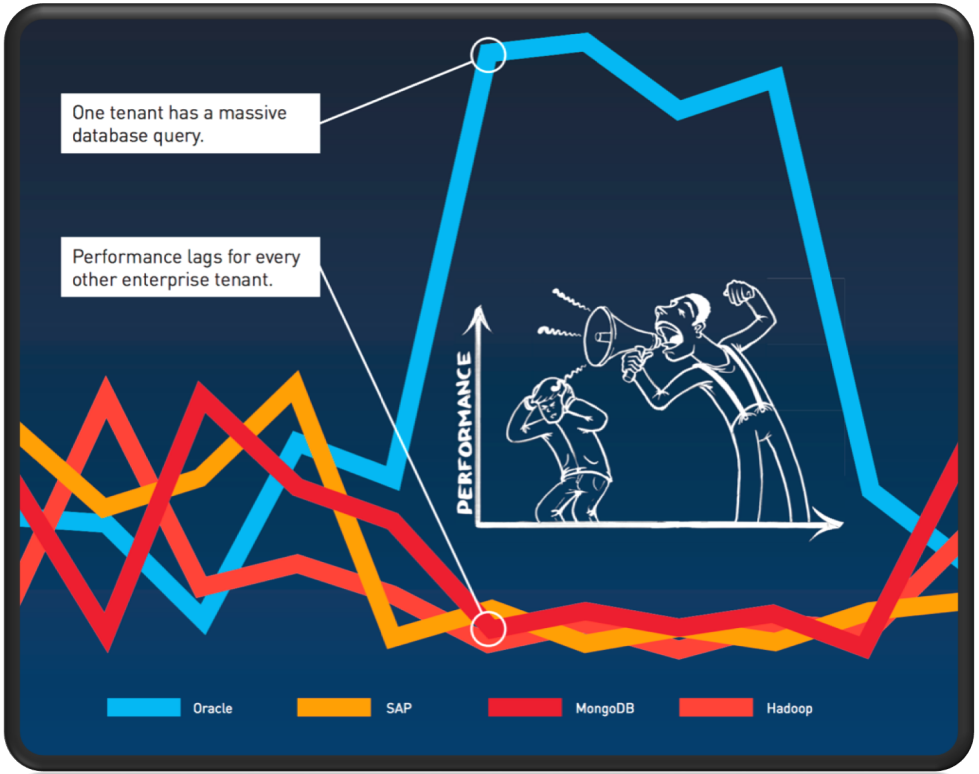


application in the “cloud” are subject to uncontrolled noise…

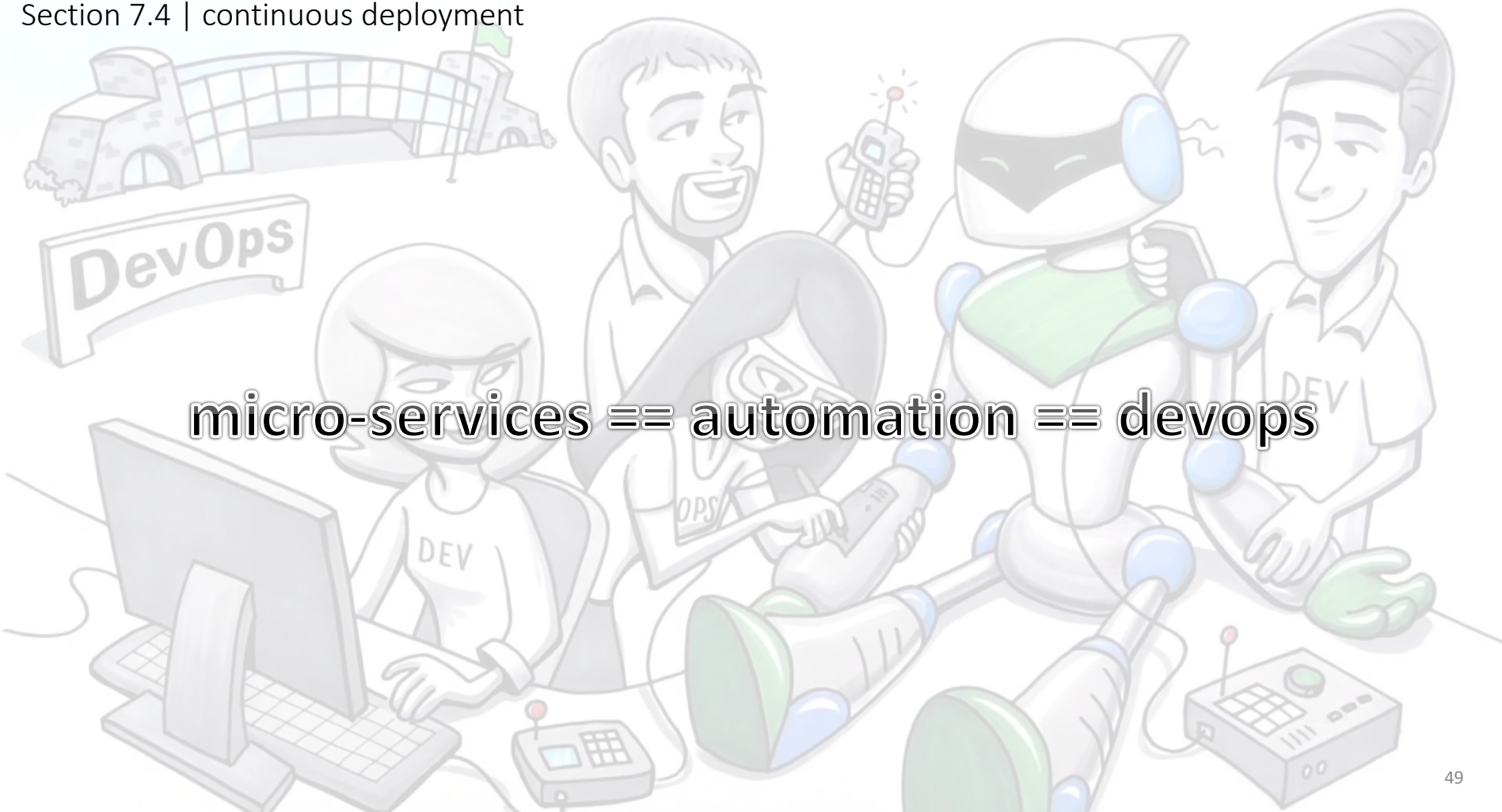
…a CaaS can amplify that noise significantly

(source) noisy neighbors

- limited visibility
- limited control
- difficult to measure

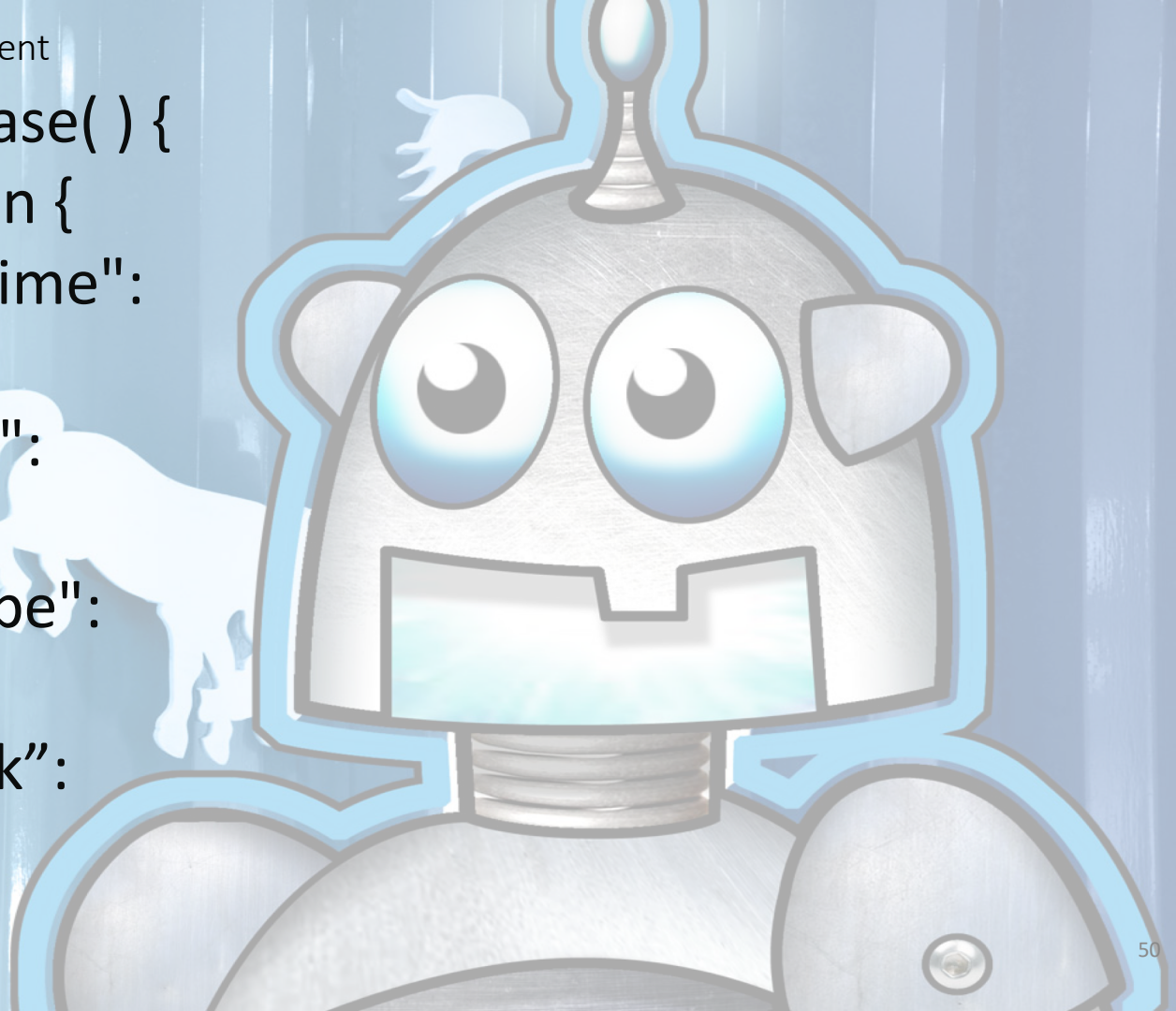


Section 7.4 | continuous deployment

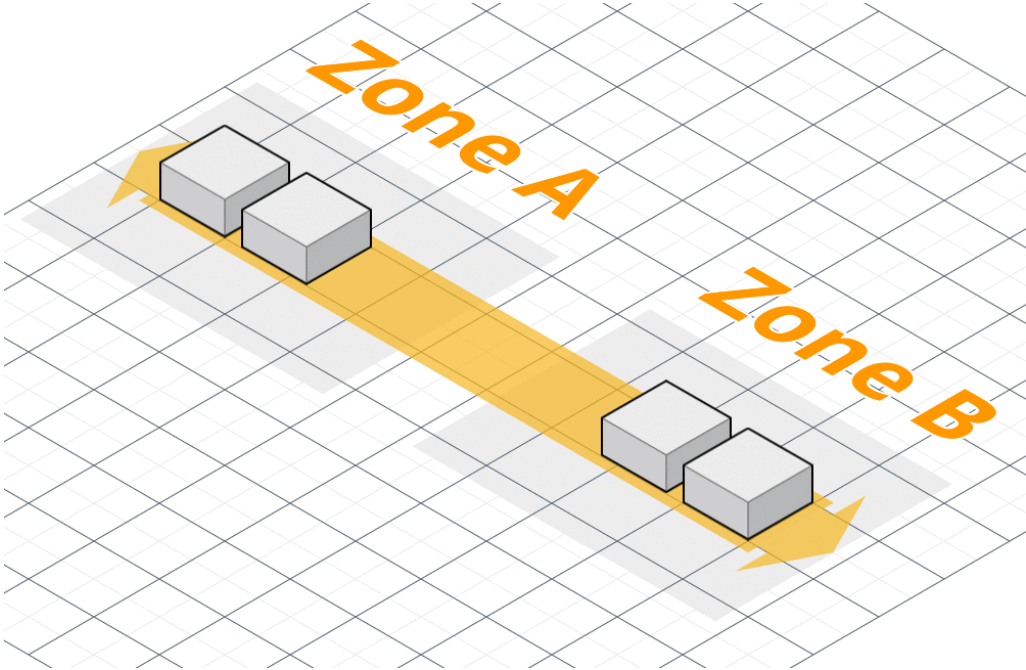


micro-services == automation == devops

```
func rolling_release( ) {
  switch do_action {
  case ”no downtime":

  case ”scheduler":

  case ”deploy-type":

  default ”rollback”:
  }
}
```



func rolling_release( )

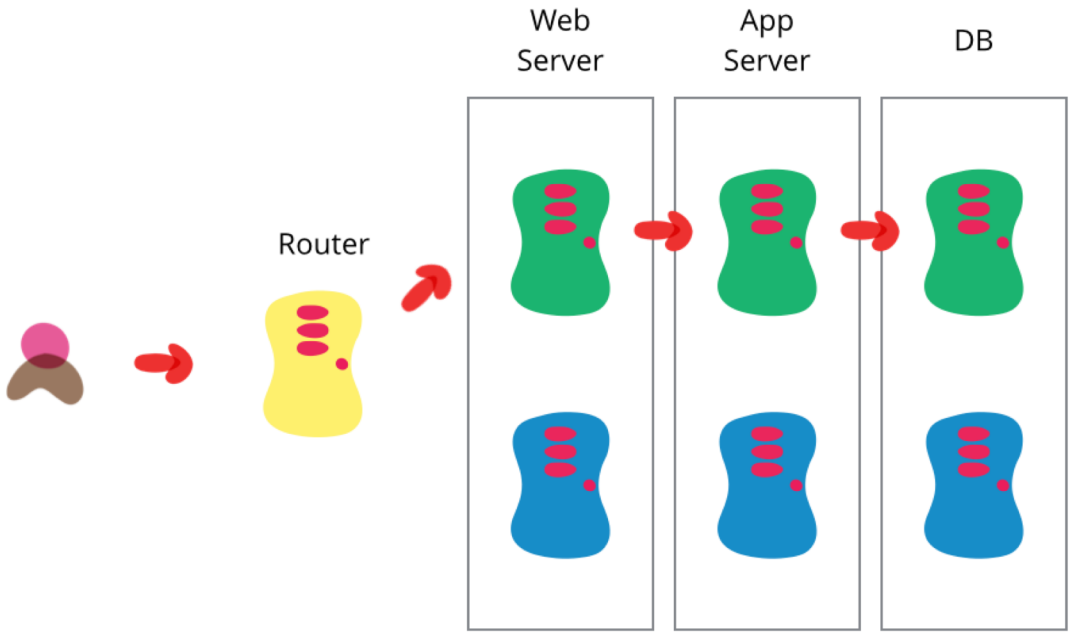


NEBULAWORKS

version release windows

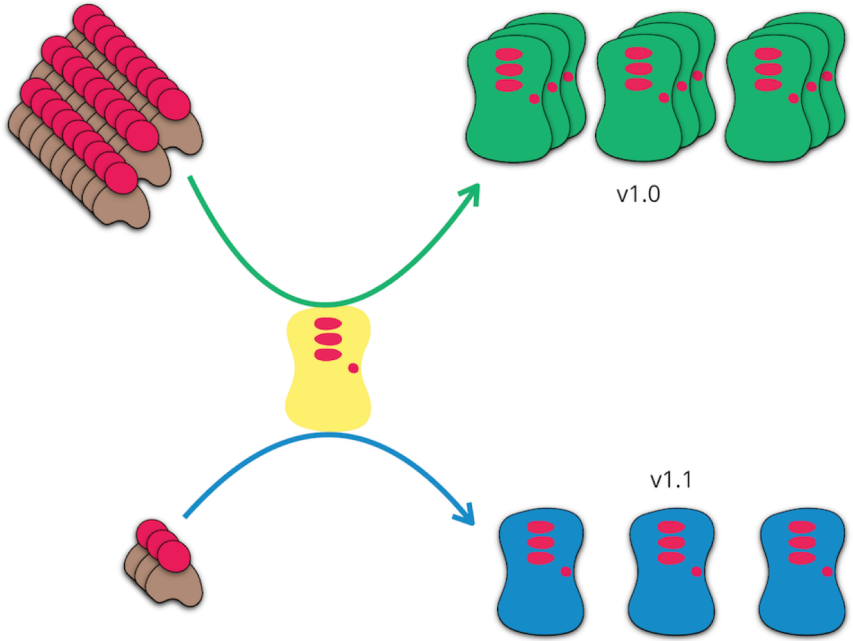
blue green

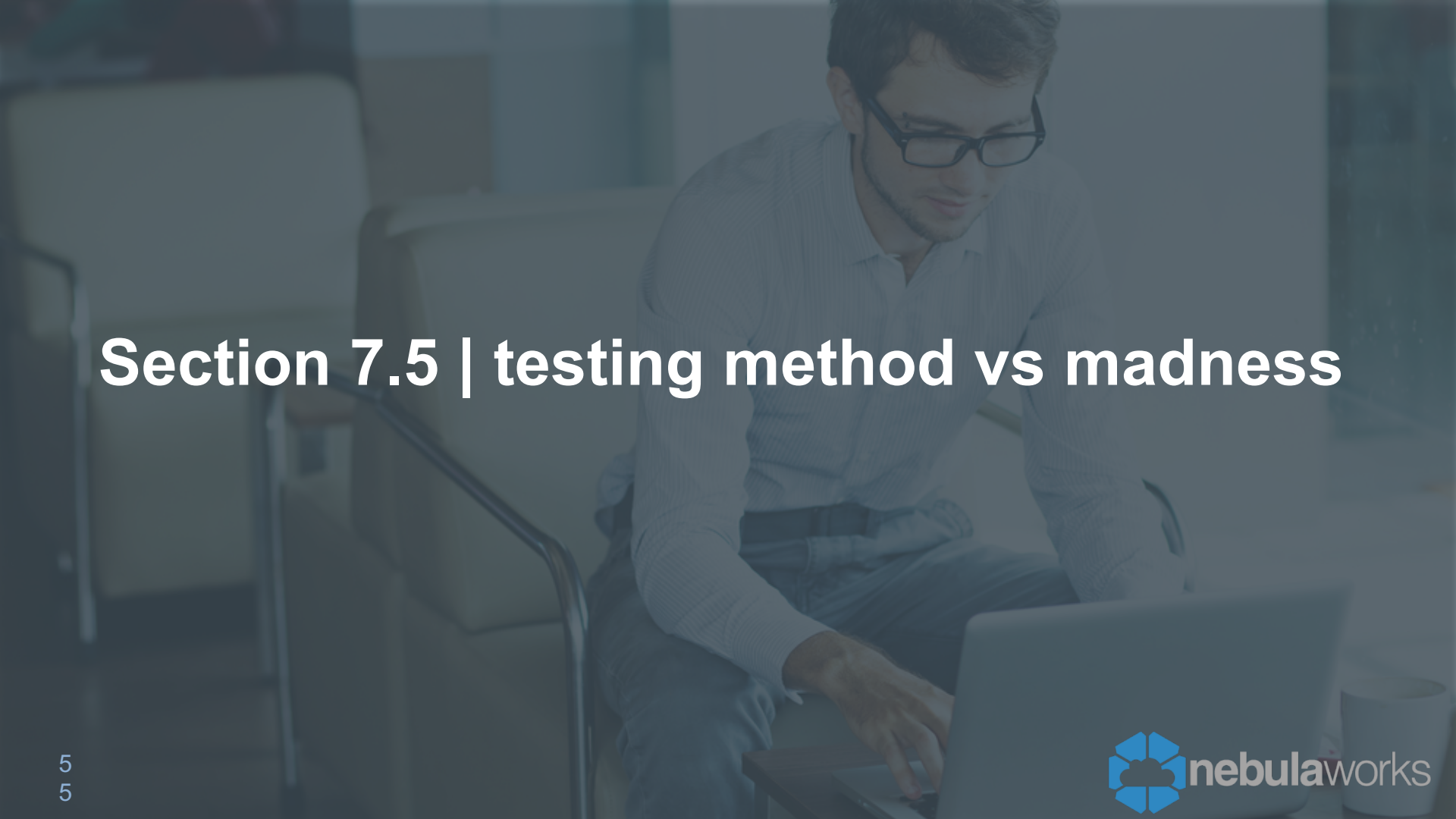
- avoid in place upgrades
- Favor immutable infrastructure
- easy rollback plan
- Switch router to the idle environment once it has been tested and deployed



canary

- great for experimentation of new features
- blue/green +
- great pre-test of rolling updates





Section 7.5 | testing method vs madness

NETFLIX

systematic vs. specific risk

- systems must be tested as a whole
- plan in failure
- unknown unknowns

feedback systems

- bottlenecks vs. failures
- events vs. metrics
- don’t forget inspection

Survey and Feedback

It is critical to our success as trainers to hear from you and get feedback on how we are doing. (It is also required by your employers. This is how we get graded on our work.)

- We welcome constructive criticism and feedback, and pay very close attention to your comments and suggestions. (In fact, these help shape and improve our courses, and comments from previous students have been factored in to improve the course you are taking today.)
- We're setting aside 15 minutes of class time for you to complete this before lunch, though it should actually take much less.
- Being satisfied, yet silent, actually hurts us. Please take a few minutes now to complete the survey.
- Thank you!

Section 7.6 | lab

Section 7.6 | lab

lab objectives

- Deploy a CaaS in a Public Cloud
- Launch an containerized app
- Checkout a container registry

NEBULAWORKS

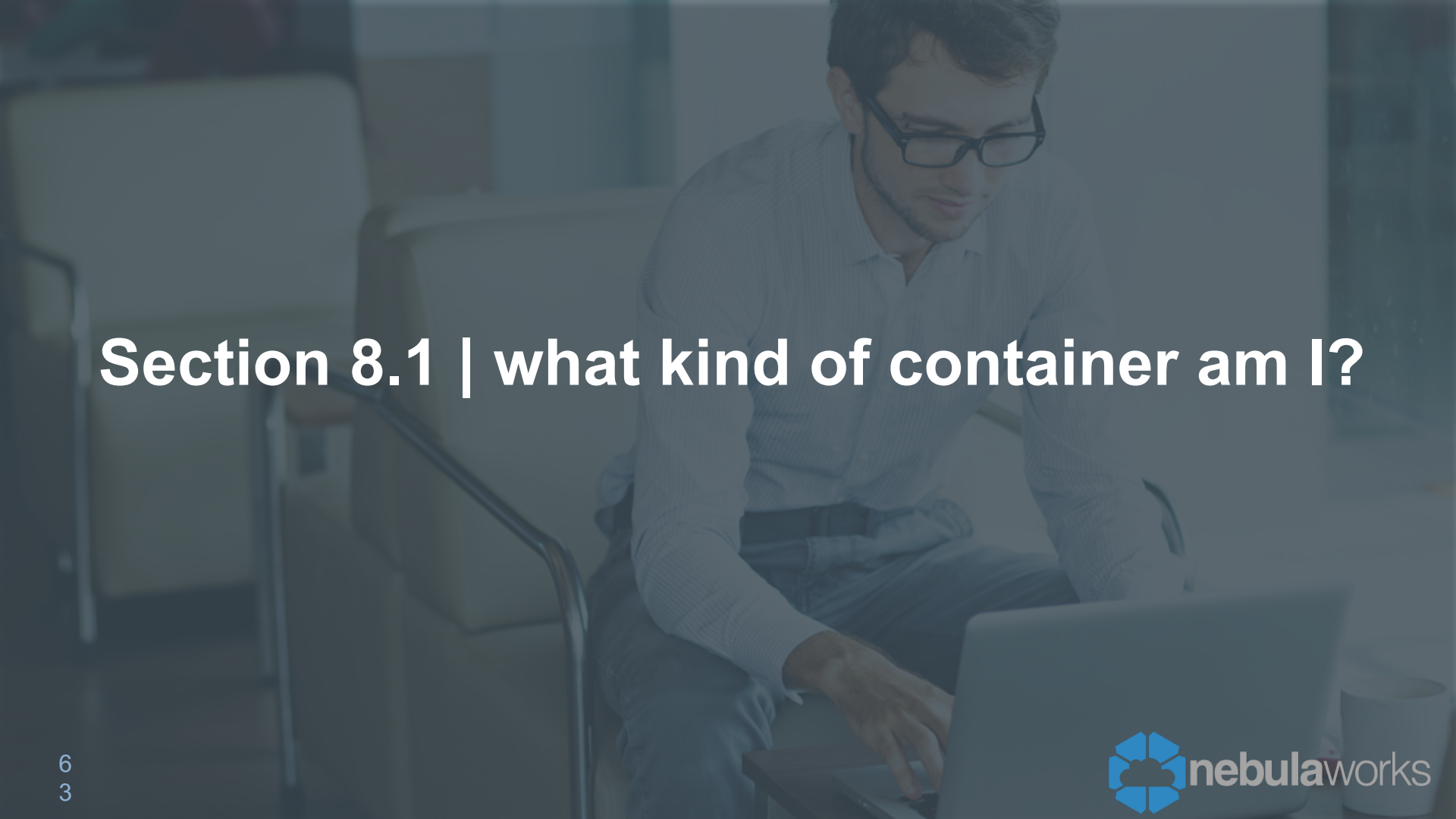
Section 7.6 | lab

lab 7 objectives

- Create a Kubernetes cluster
- Launch an containerized app

Embracing a Platform and an Opinion

module 08



Section 8.1 | what kind of container am I?

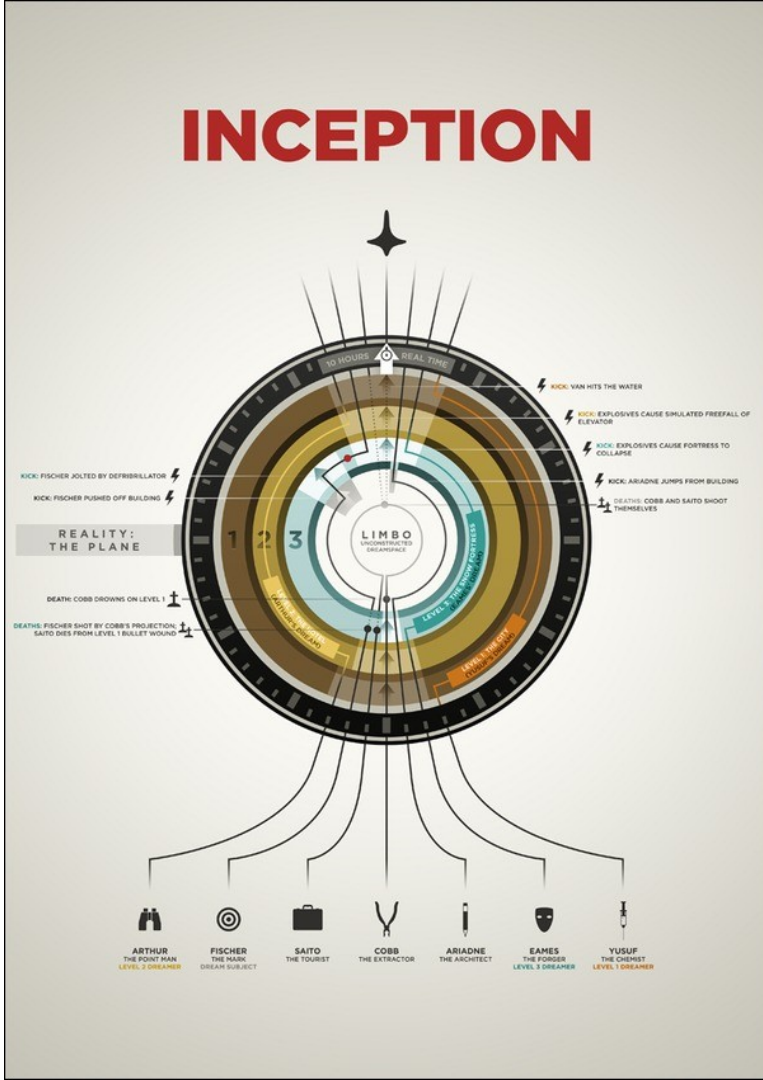
ultimately a container is merely a process or set of processes running within a shared kernel. the design approach used to build a given container will be one of the most deterministic factors in defining the cost against the system.

full system container

- no CoW fs
- easier to turn into pets
- system vs app oriented

virtual process container

- app vs. system oriented
- generally uses CoW fs
- closer to cattle



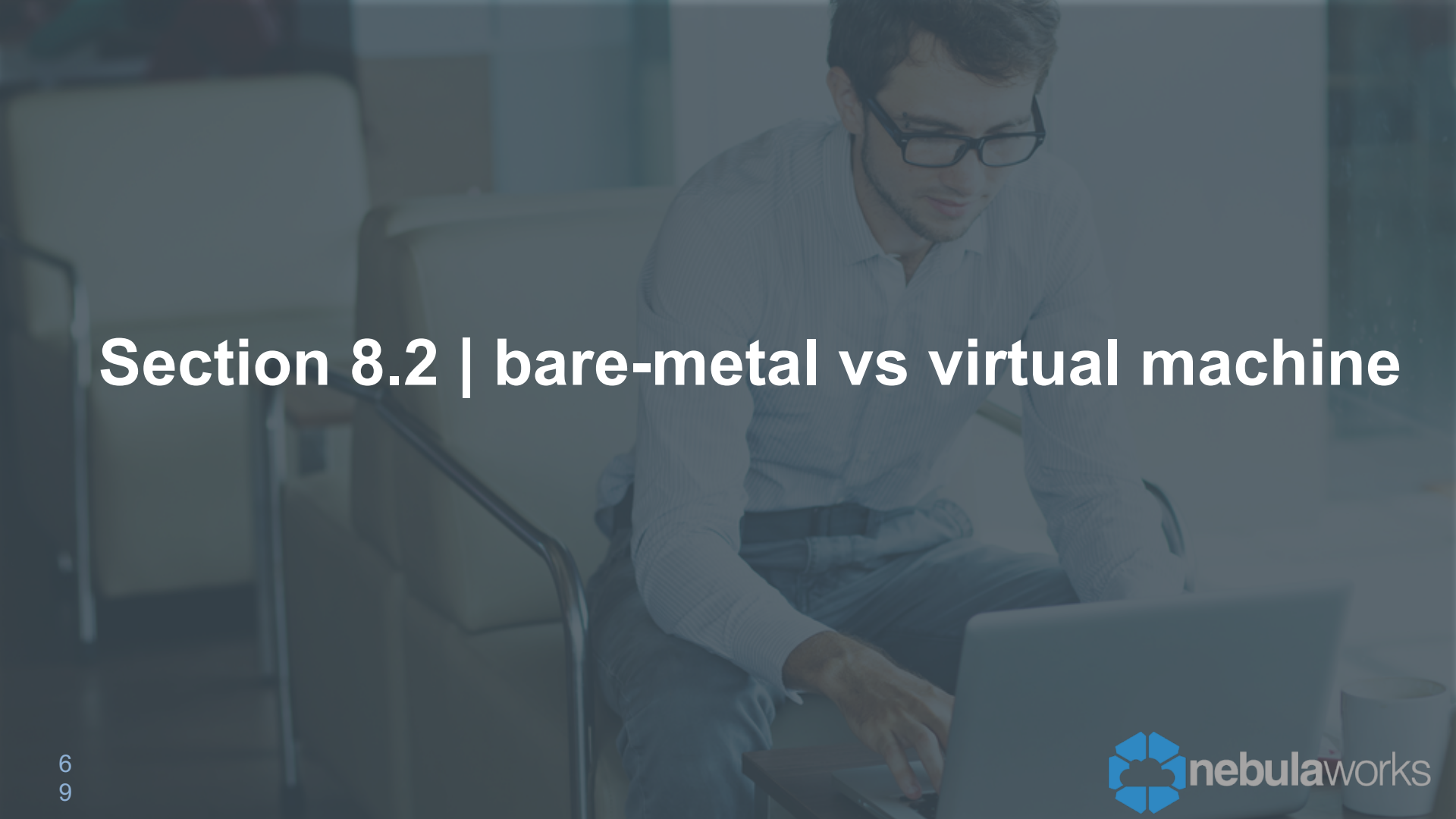
NEBULAWORKS

app container

- super lean
- micro-service friendly
- luvs pods!

docker container

- is there any other kind
- dev friendly
- highest compatibility



Section 8.2 | bare-metal vs virtual machine

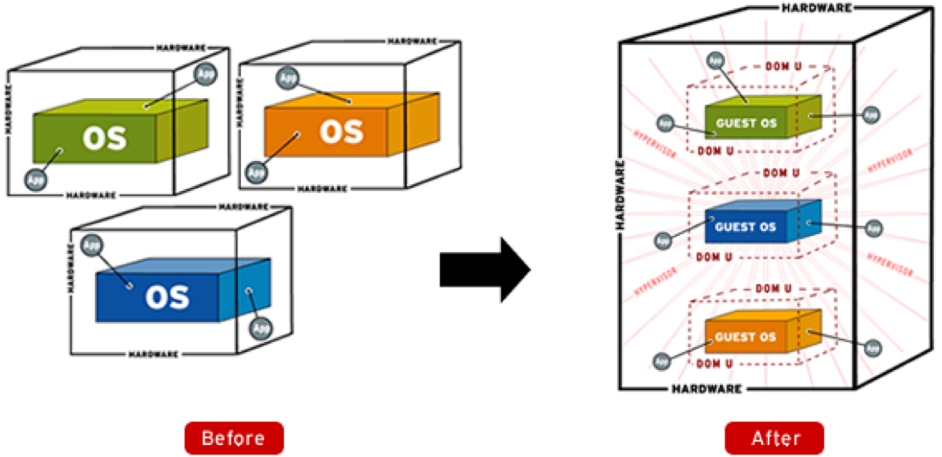
since containerization represents another kind of virtualization relative to machine virtualization then the choice of the underlying infrastructure used in building the host for the container engine will also impose costs. these can generally be categorized as systematic or environmental constraints.

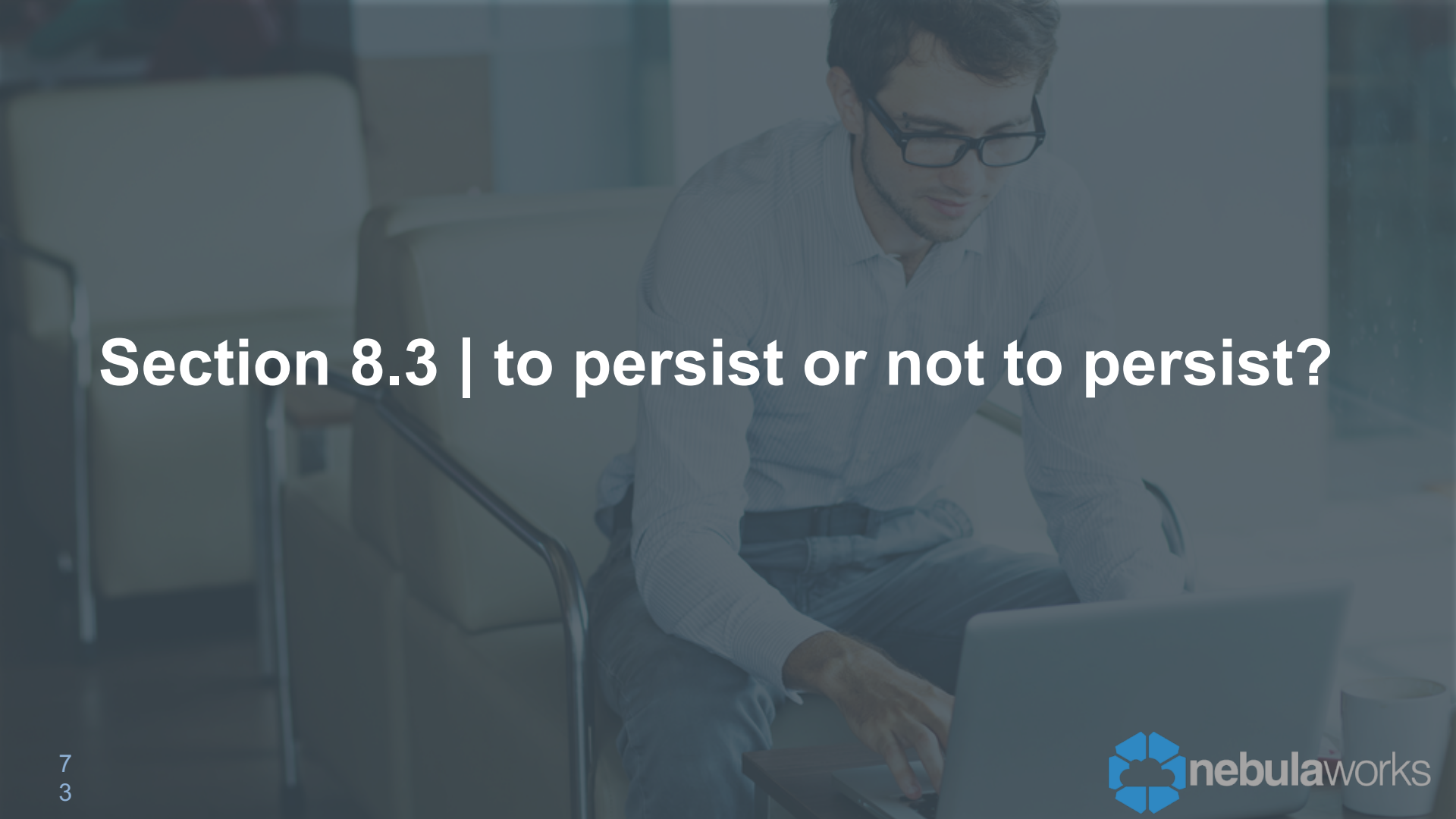
bare-metal

- strengths
  - performance
  - simplicity
  - utilization
- weaknesses
  - expense
  - multi-tenancy
  - management

virtual machine

- strengths
  - multi-tenancy
  - management
  - expense
- weaknesses
  - performance
  - complexity





Section 8.3 | to persist or not to persist?

most large scale production environments deploying containers at scale treat the artifact largely as a short lived one. as such the question of persistence must be addressed to determine the limits that surround density and possibly scalability.

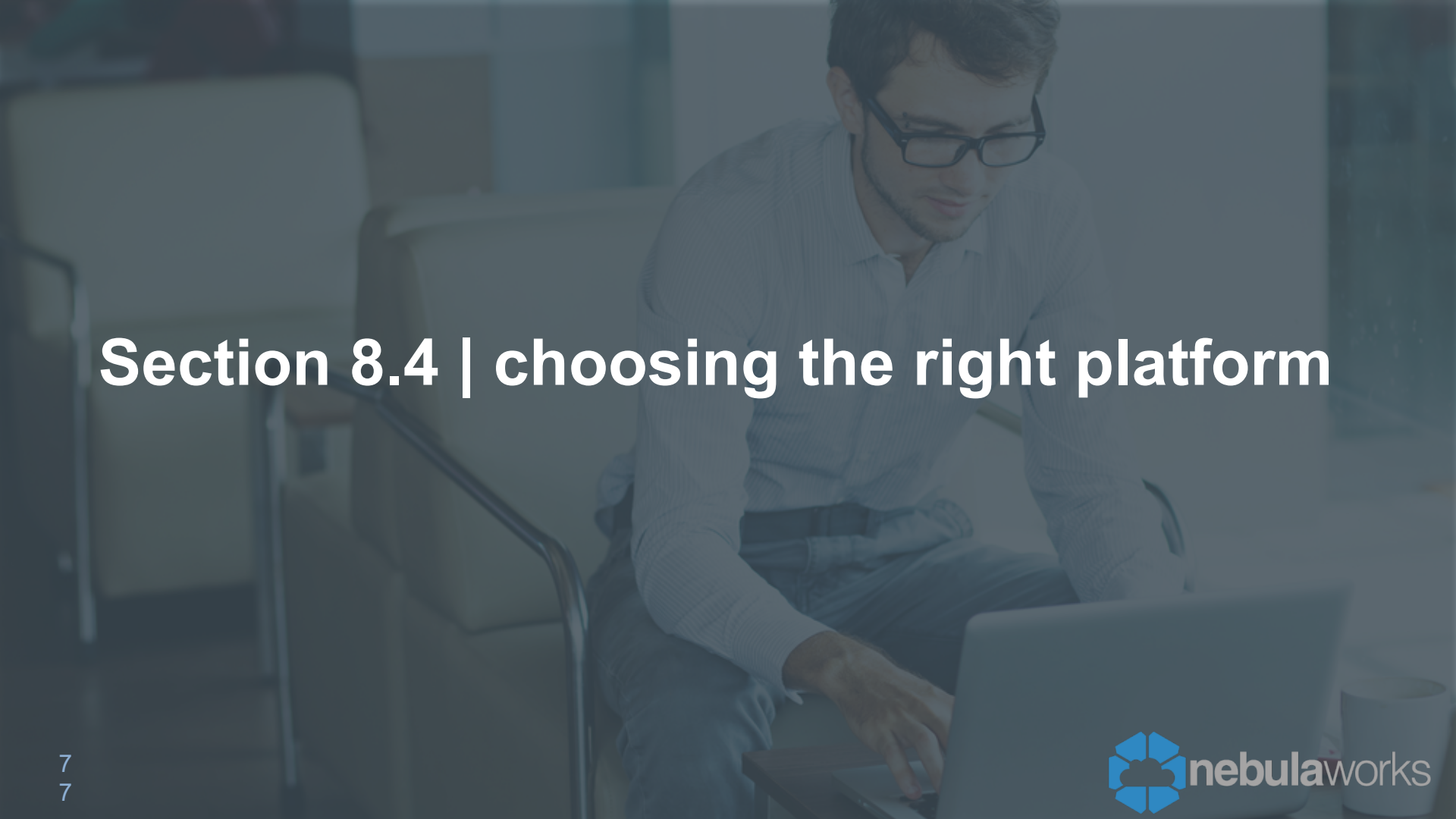
"This time I think it's a hardware problem."

as a network service

- persistent
- defined as an infrastructure service
- delivered by container?

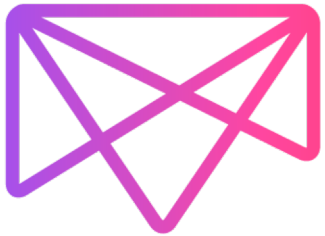
as a part of the artifact

- to be or not to be ephemeral?
- how big is that artifact?
- sensitivity



Section 8.4 | choosing the right platform

mesosphere



- Oldest Platform (at least at the core)
- Reaches beyond Containers
- Multiple schedulers available
- Container management options
- Highly scalable

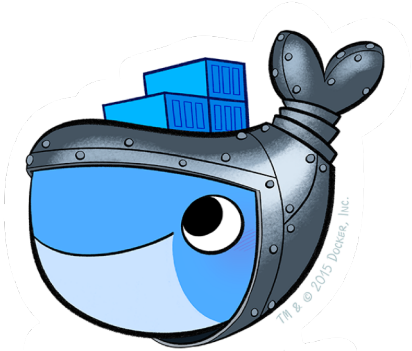
kubernetes

- Backed by Google
- Container centric
- Complete SaaS Platform
- Container management options
- Adoption by PaaS platforms

Section 8.4 | lab

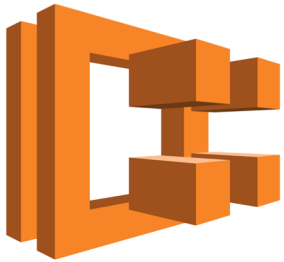
nebulaworks

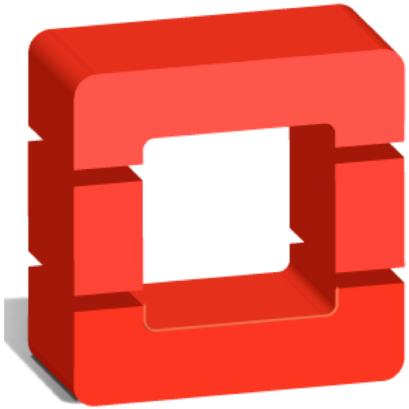
Docker Enterprise Edition



- Docker native platform
- Looks end to end at the software cycle
- Relatively easy to use for novices
- Most widely tested Container Engine
- Available on most public clouds

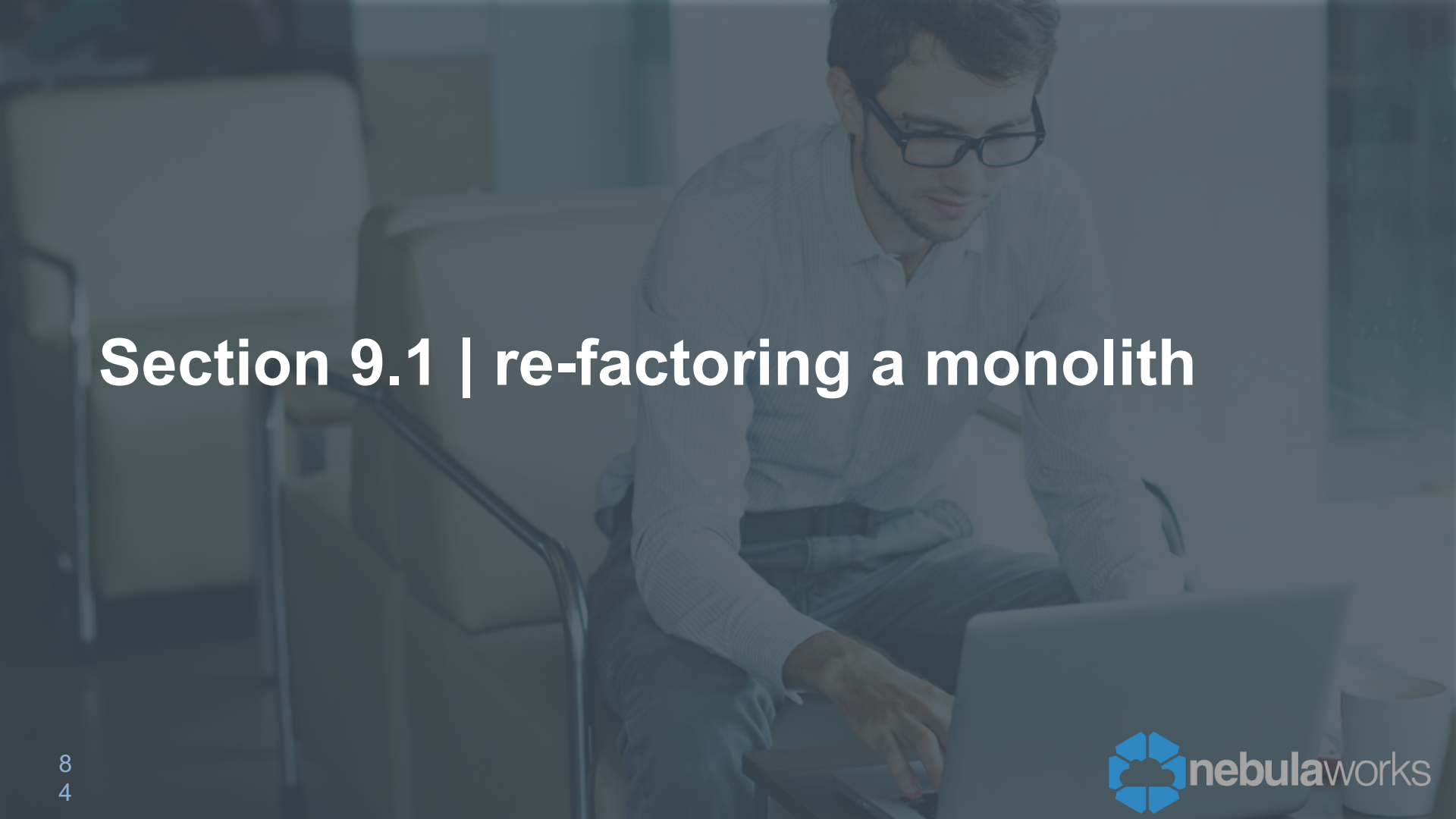
others





TROUBLESHOOTING CONTAINERS

module 09



Section 9.1 | re-factoring a monolith

many enterprises want to use containers like “super” VMs …
so they start the containerization process as another form of p2v migration

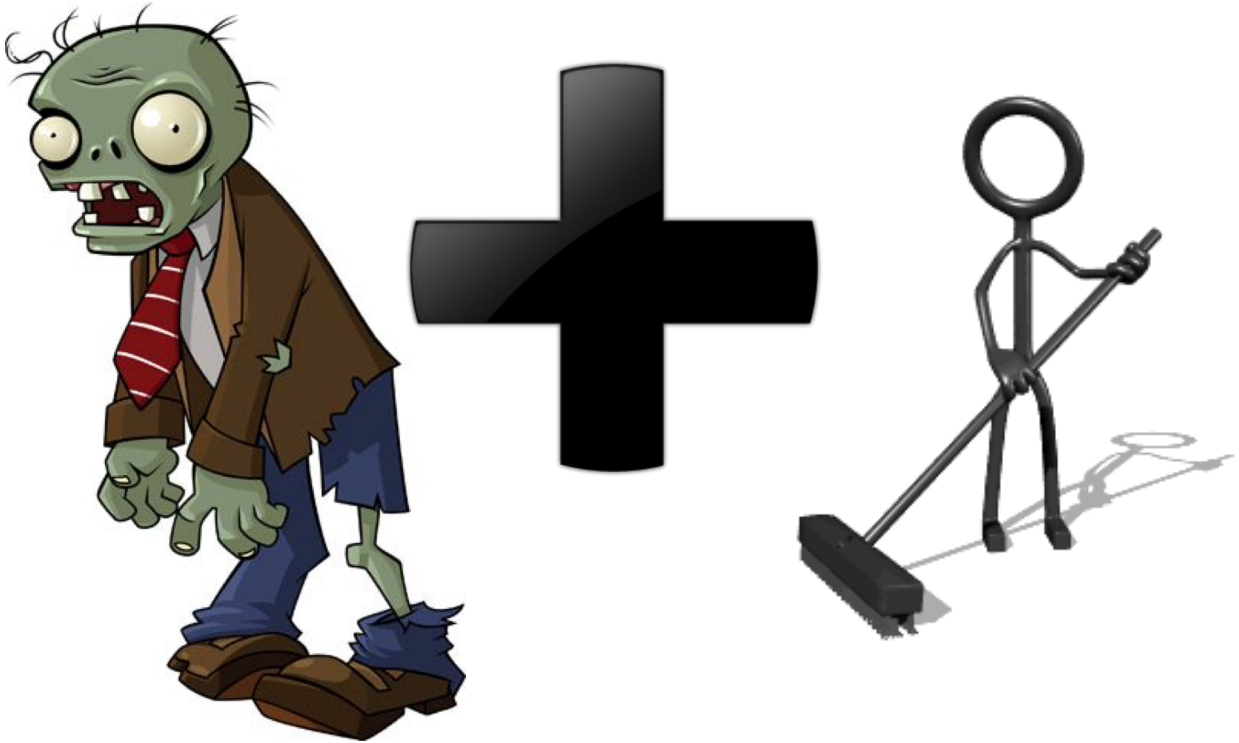
base images are not AMIs (Amazon Machine Images)

- fewer default binaries
- don’t have service managers
- shares a kernel

the plumbing is different

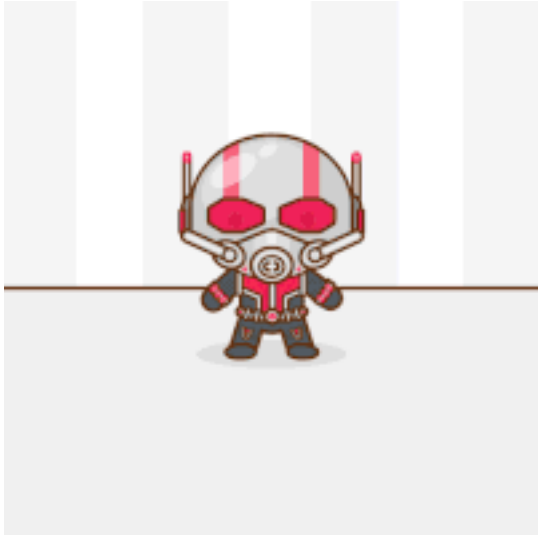
- did someone say networks
- Sometimes limited abilities to tweak things

who's in charge of running the show and clean up >>>



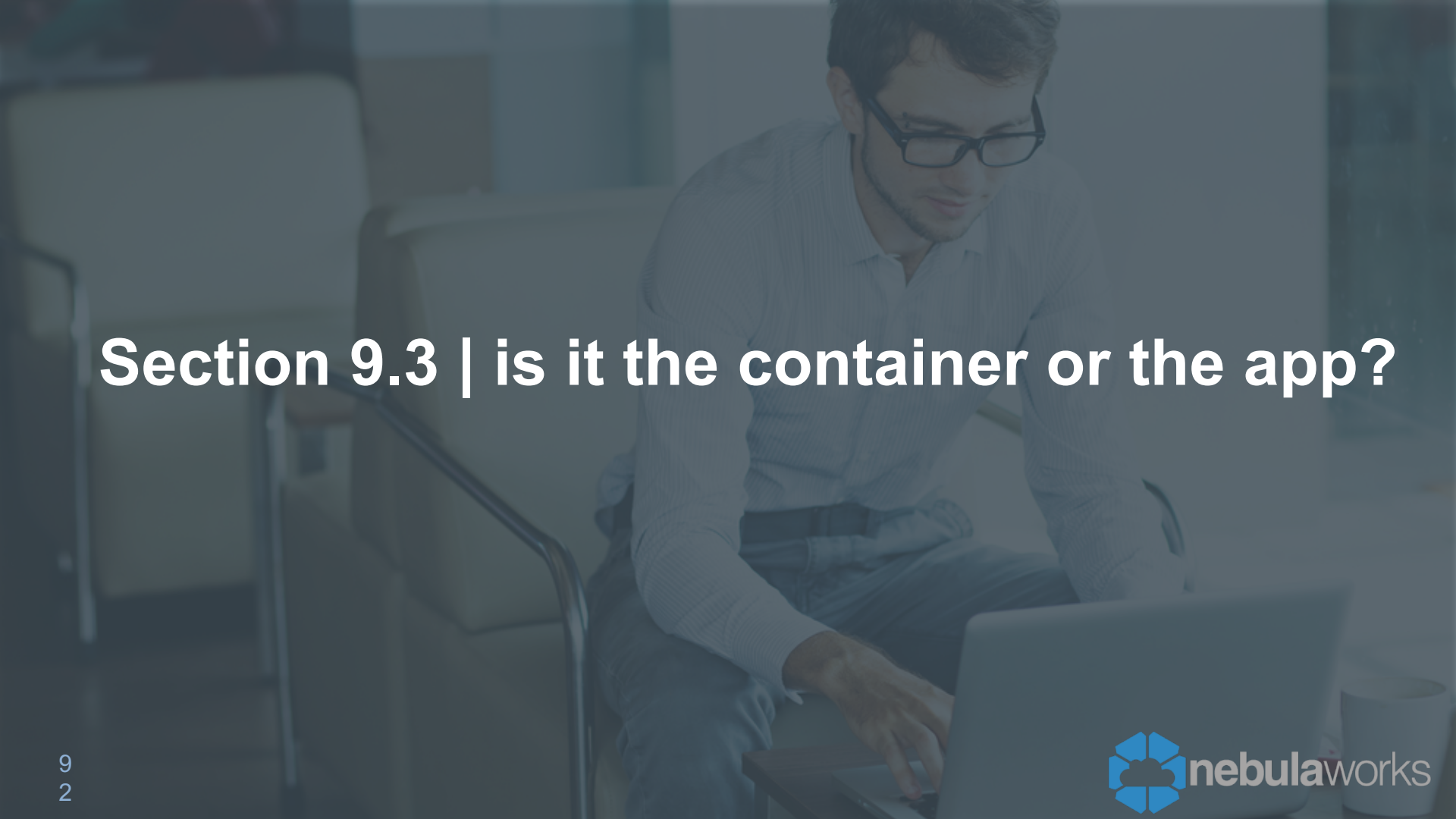
Section 9.2 | micro-service ready?

countless developers have now declared that they are "going" micro-service.

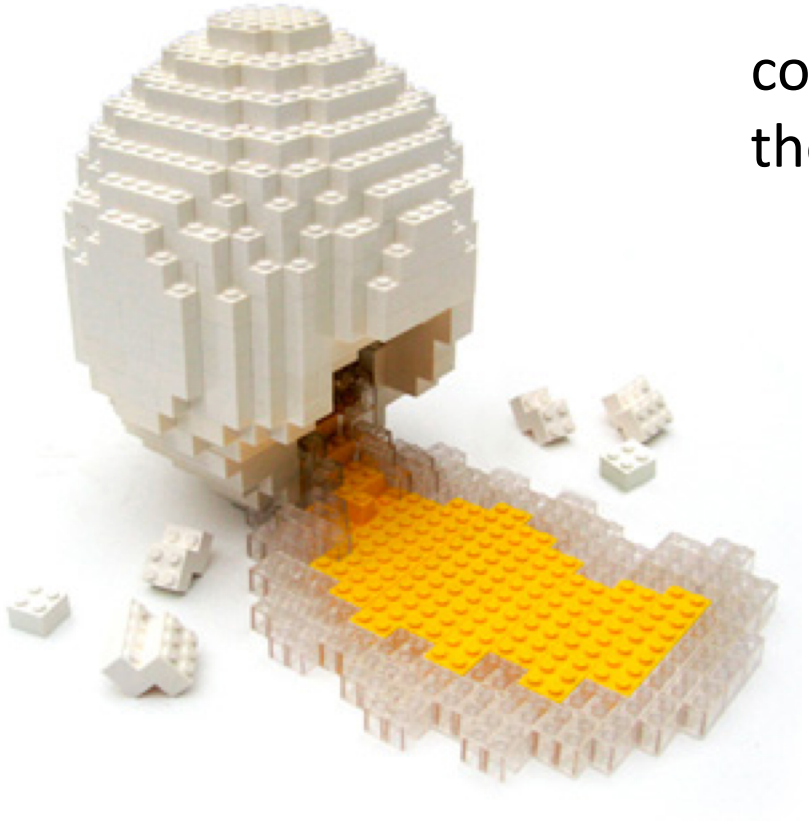


are tools available to make good determinations

- meta data services?
- responsive to environment
- health checks?



Section 9.3 | is it the container or the app?



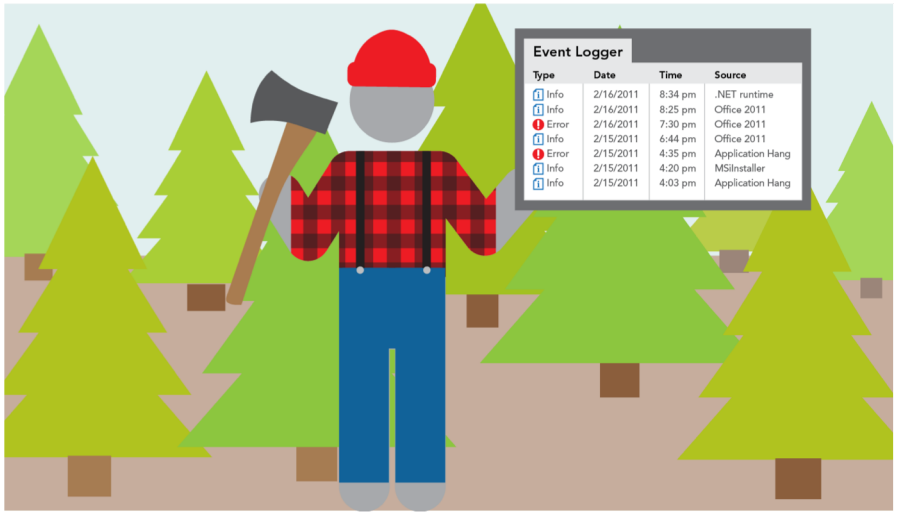
containers are the new lego in the box…

…making it the source of all new errors

which logs confirm your view of the problem

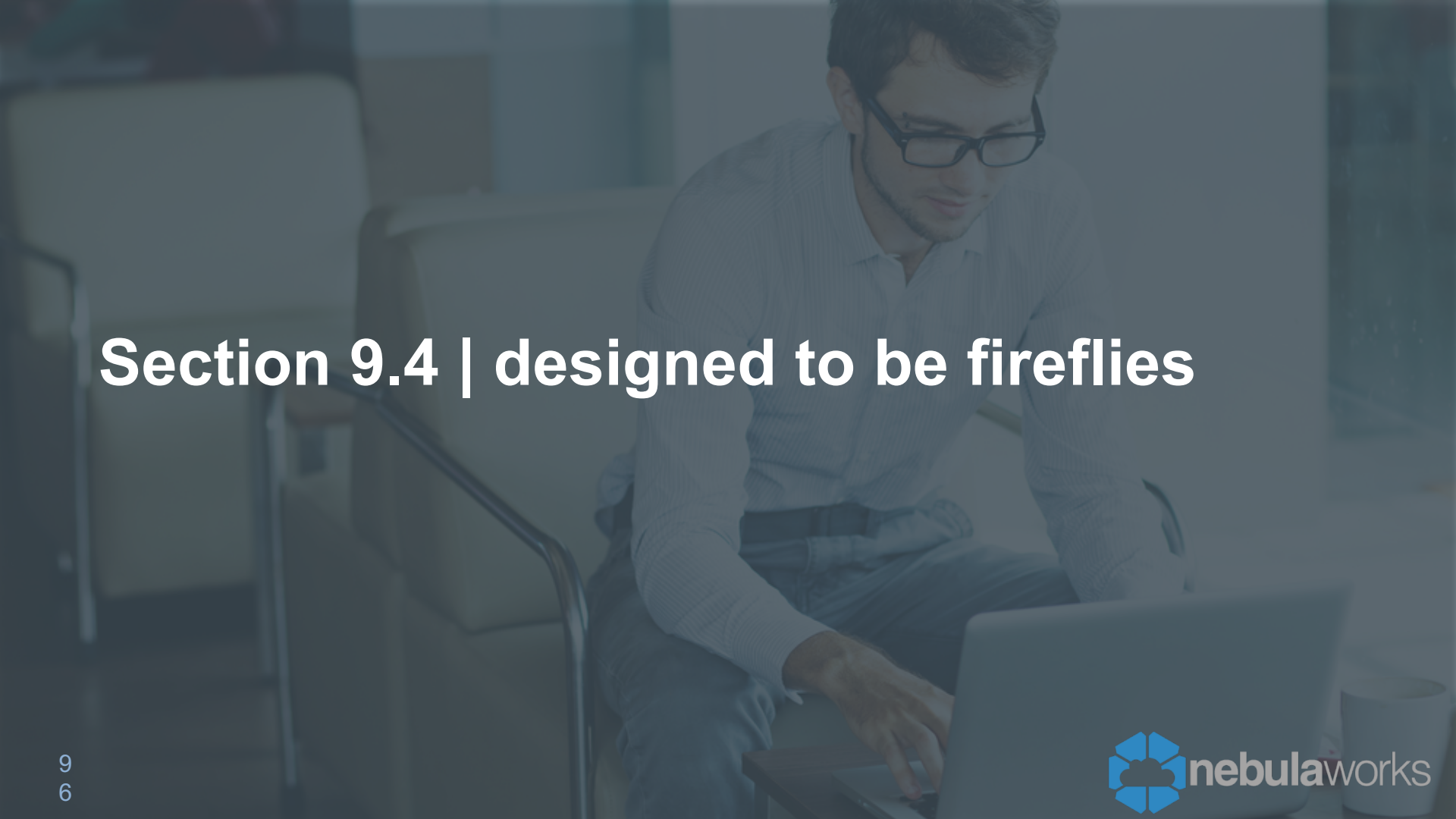
three-types

- daemon logs
- container logs
- application logs



how do you test conclusions

- debug/test container
- finding the root causes
- repeatable processes



Section 9.4 | designed to be fireflies

“beautiful things never last that's why fireflies flash” – *Ron Pope*

The average lifespan of a container in the cloud is ¼ that of an instance.

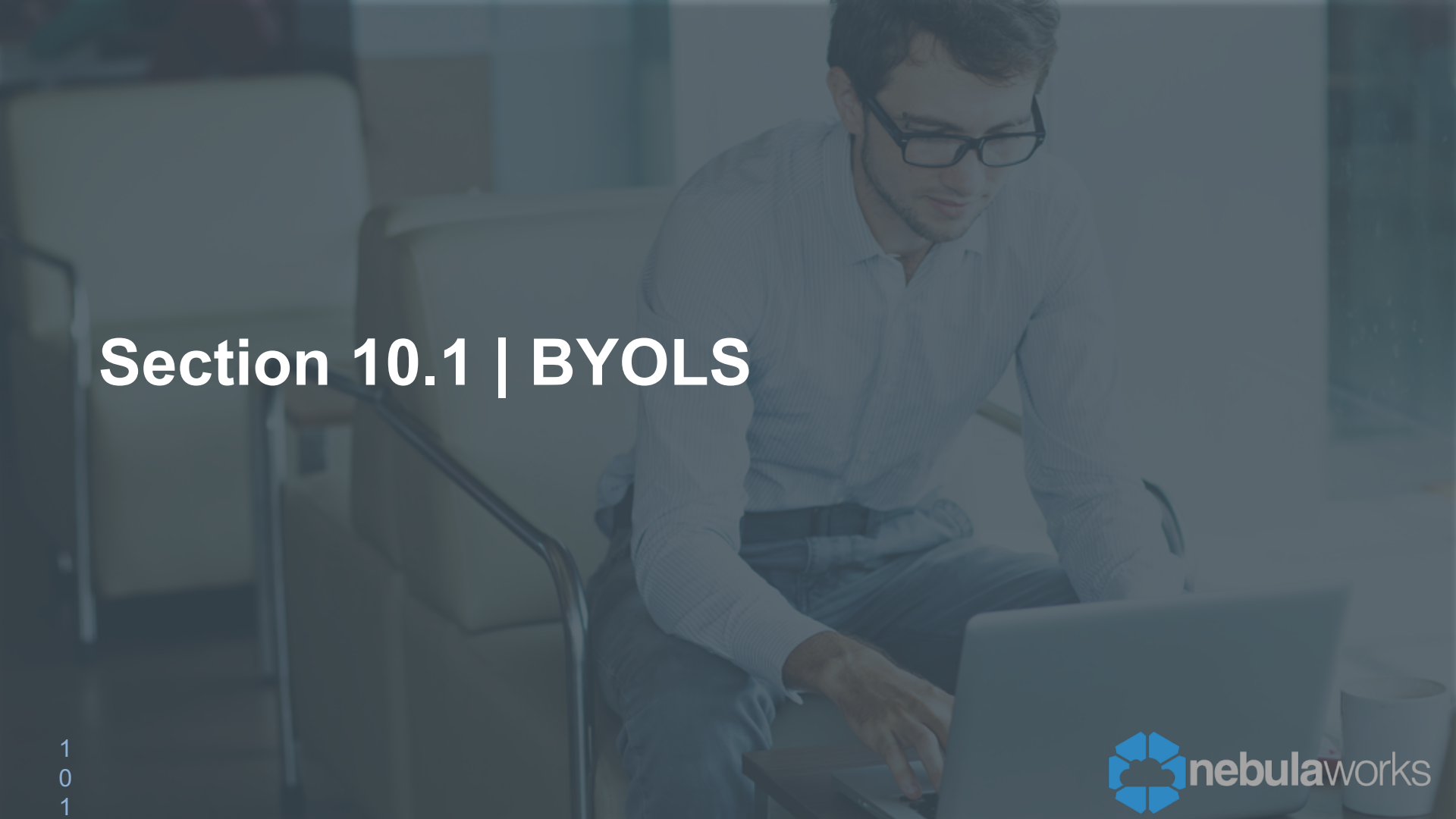
generally scaling is designed in

problems of garbage and zombies

MONITORING CONTAINER SERVICES

Module 10

nebulaworks



Section 10.1 | BYOLS

most existing infrastructures have logging systems in place and so there is a desire to bring containers into the fold. So when you Bring Your Own Logging System the two things to consider is how to plug containers into them and since often containers represent discrete processes what kind of metrics or events need collecting

* docker engine plugin architecture for logging systems
* alternatives to direct engine integration
* what are the right optics

docker engine plugin architecture for logging systems

key facts >>>

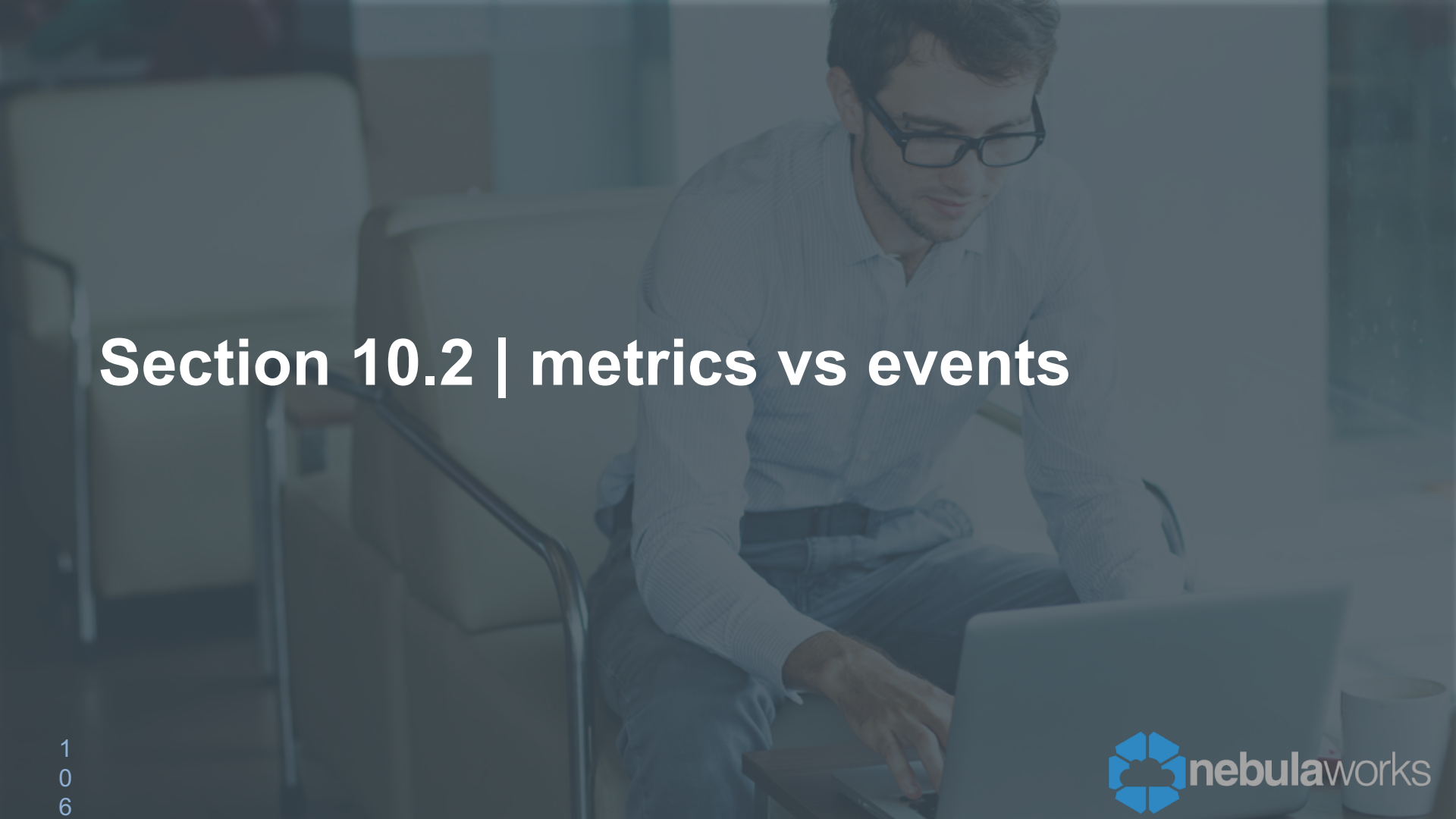
- supported at the daemon level
- docker logs only works with json-file and journald
- native options: json-file, syslog, journald, etwlogs (Event Tracing in Windows)
- external options: gelf, fluentd, awslogs, splunk, gcplogs

alternatives to direct engine integration

- system level logs
  - look for systems that monitor process events and capture machine data
- mounted logs
  - containers have logging services
  - send output to volume mount
- third party platforms

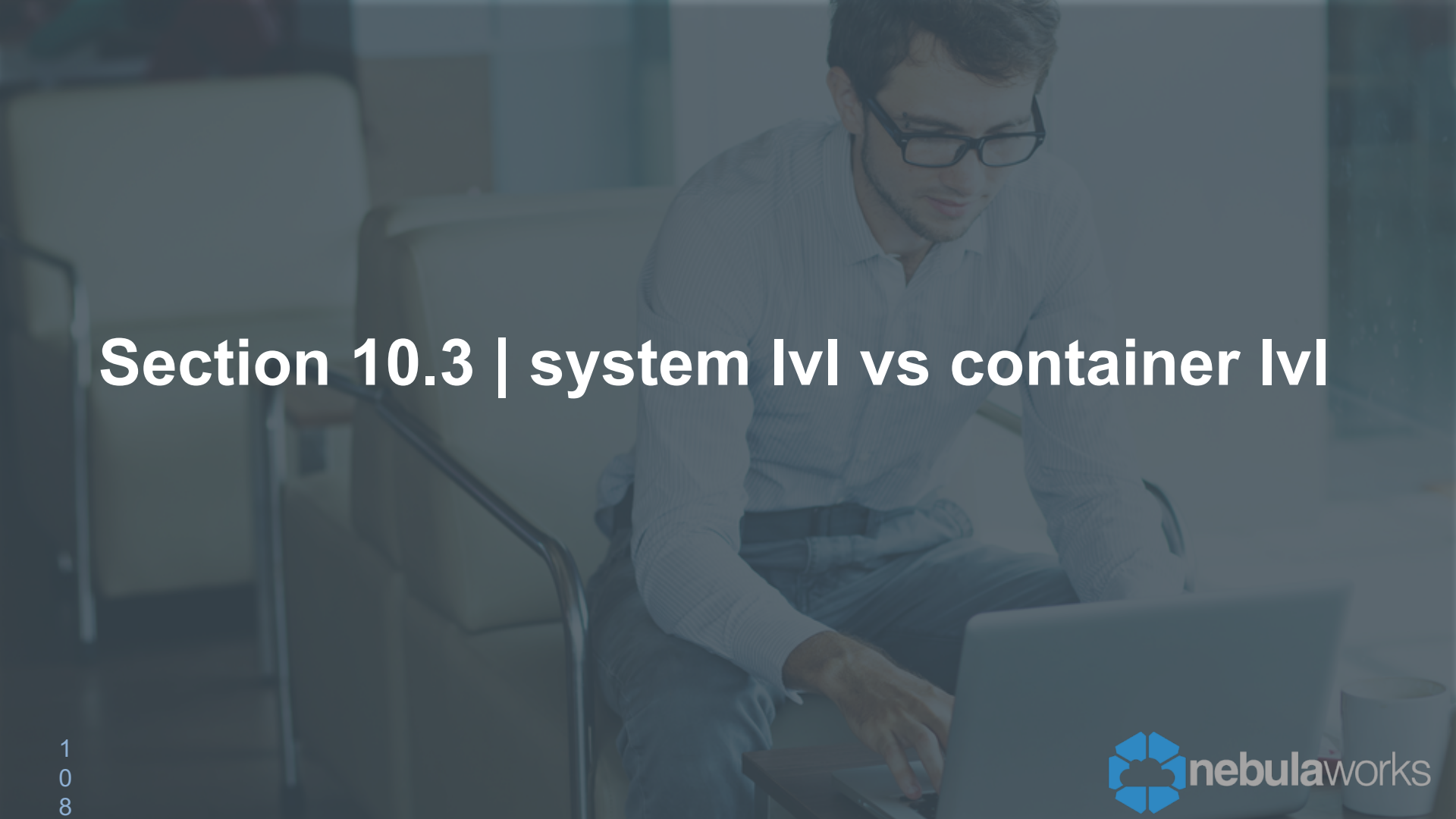
what are the right optics

- types of machine data
  - metrics – performance, reliability, load
  - events – application level, system level, infrastructure level
  - stack traces – application level call traces



Section 10.2 | metrics vs events

if the container is the new artifact and continuous deployment the new pipeline it seems obvious what's important now to watch and collect. ultimately it is the notion of predicting and managing the health of an application that will drive the selection criteria



Section 10.3 | system lvl vs container lvl

when systems are condensed to container management system where do we plugin to get the information we need? collection services must find a ways of not only gathering the information but organizing it to close the right feedback systems

Section 10.4 | ecosystem players

nebulaworks

so who's providing the monitoring systems need to tame the CaaS?

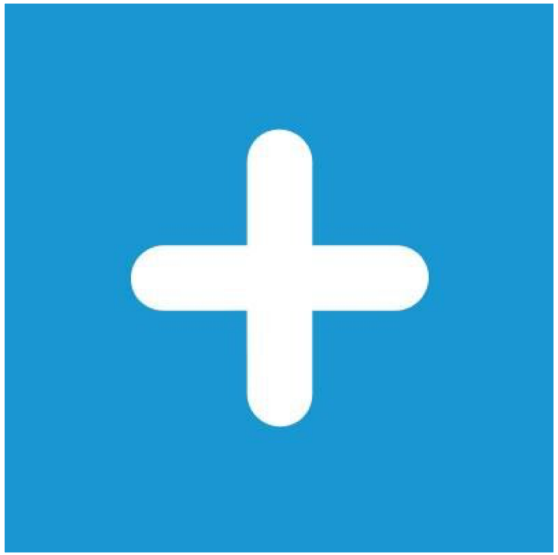
- Datadog
- Sumologic
- New Relic
- CoScale
- etc

datadog key features & facts

- founded in 2010
- SaaS based model
- custom interactive dashboards
- provides API for integrations
- provides event and metrics correlation
- https://www.datadoghq.com/

sumologic key features & facts

- founded in 2010
- SaaS based model
- real-time log analytics
- uses machine learning
- agent based
- https://www.sumologic.com/

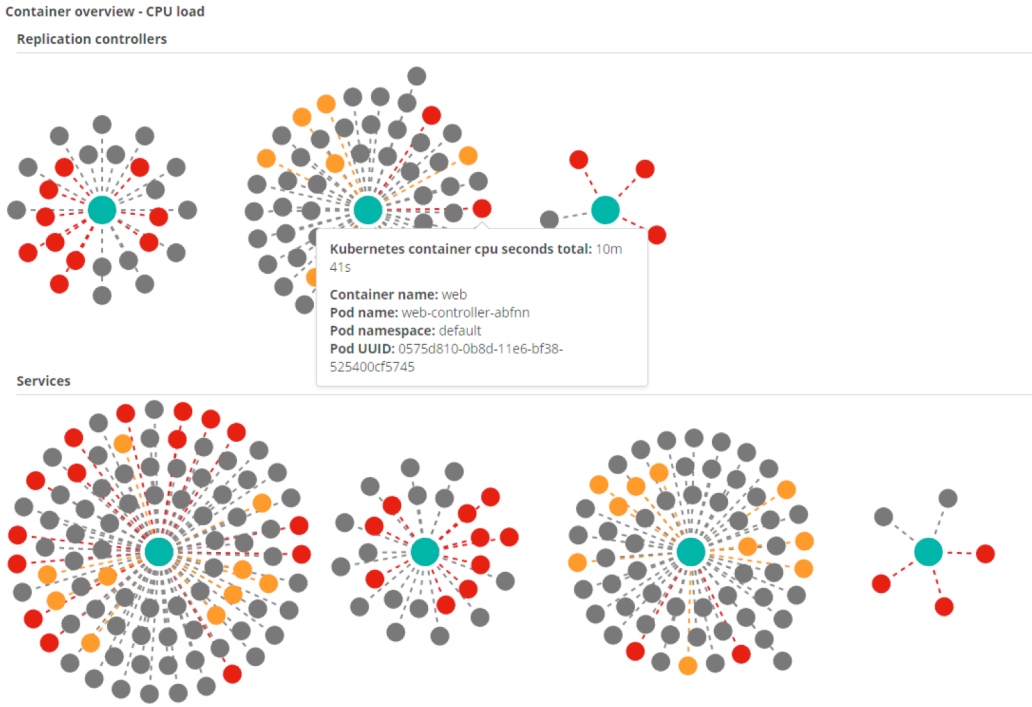


new relic key features & facts

- founded in 2008
- SaaS based model
- application monitoring
  - end-to-end transaction tracing
  - code-level visibility
- synthetic monitoring
- iot/mobile monitoring
- https://newrelic.com/

CoScale key features & facts

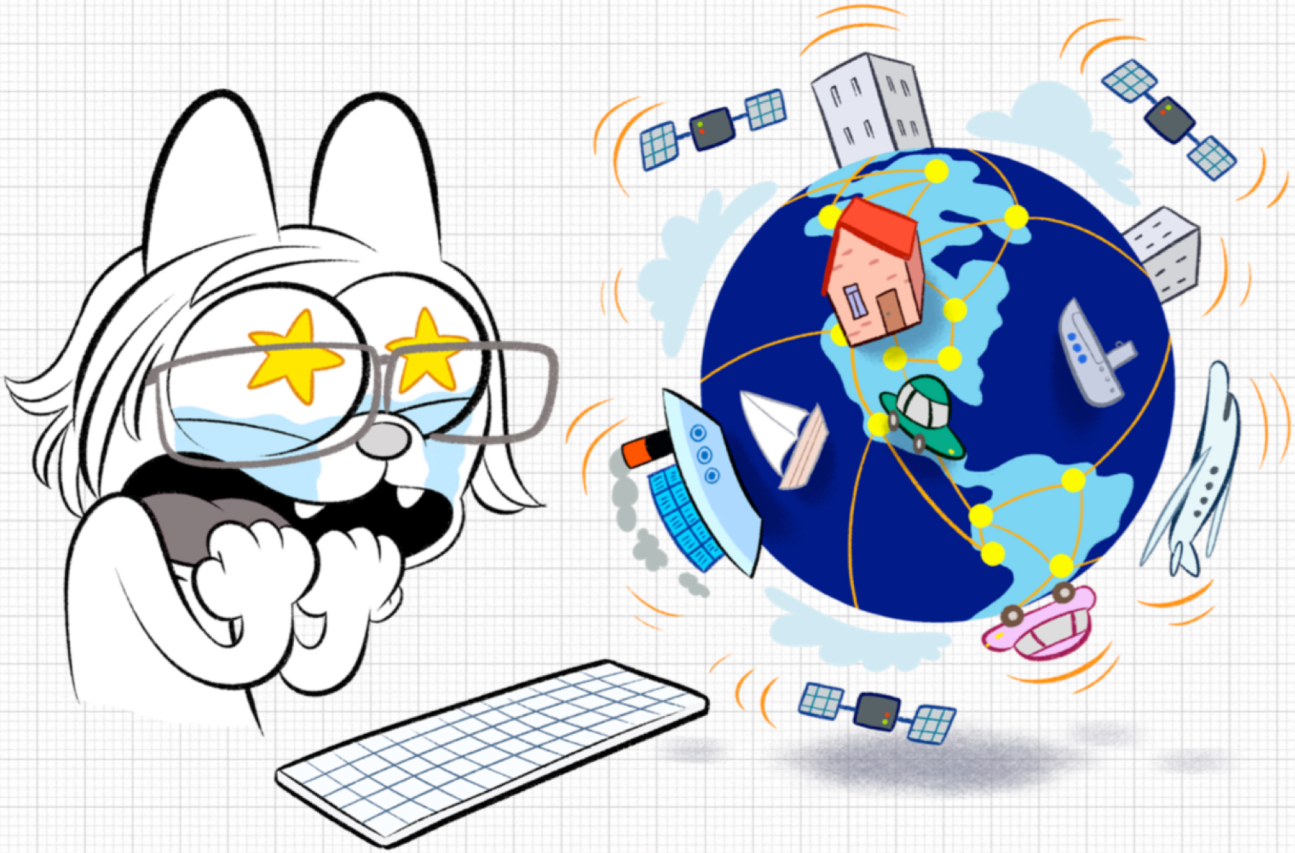
- Founded ~2011
- Built on a widely-integrated platform
- Application monitoring
  - Optimized for containers and microservices
  - Container-level visibility
- Lightweight
- Automated anomaly detection
- https://coscale.com/



Section 10.5 | DIY???

nebulaworks

for the adventurous there is always the DIY approach using available OSS components to build the tailored solution to help not only monitor the obvious but one that can provide the ability to find subtle bottlenecks, work with failure and feedback systems, and forecast demand.



Thank You!

nebulaworks