DOCKER TROUBLESHOOTING & SUPPORT

HOW WE TEACH

- Docker believes in learning by doing.
- The course is lab driven:
  - Demo Exercises
  - Signature Assignments
- Work together!
- Ask questions at any time

SESSION LOGISTICS

- 2 days duration
- mostly exercises
- regular breaks

ASSUMED KNOWLEDGE AND REQUIREMENTS

- Strong knowledge of Linux
- Familiarity with Docker

YOUR LAB ENVIRONMENT

- You have been given several instances for use in exercises.
- Ask instructor for access credentials if you don't have them already.

COURSE LEARNING OBJECTIVES

By the end of this course, learners will be able to:

- Plan a troubleshooting strategy supported by information provided by the Docker platform
- Identify and fix issues with the Docker Engine, UCP and DTR
- Identify and fix system resource problems impacting the Docker Platform

PROBLEM SOLVING STRATEGIES

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Devise a specific strategy to approach a given Docker problem
- Narrow down the problem-space of a given Docker issue
- Follow Docker logs and metadata streams across related components

ROOT CAUSE

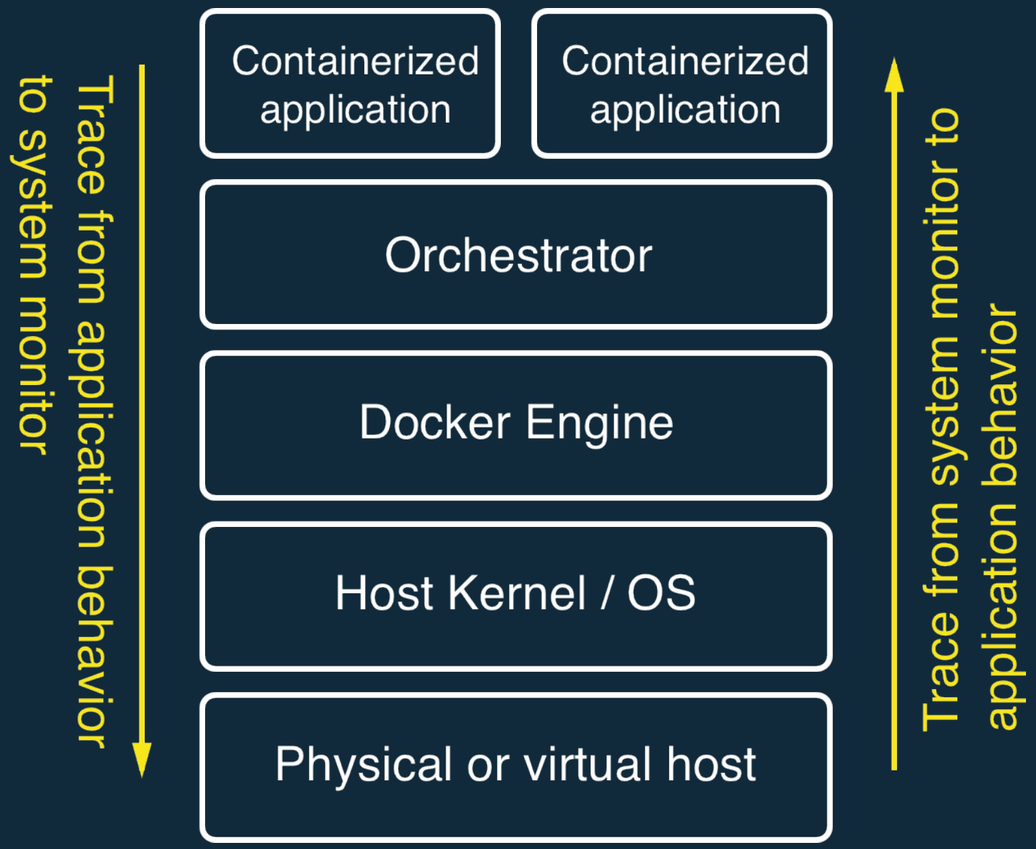
Three gross categories of root cause:

- Infrastructure
  - Network, OOM, disk...
- Docker
  - Orchestration, containerization, Docker config, EE components...
- Application
  - Bugs, image creation, app config...

INVESTIGATING RELATIONSHIPS

- Want to relate immediate problem to other observables
- Think of relationships
- Consider stacks, onions, webs, and flows

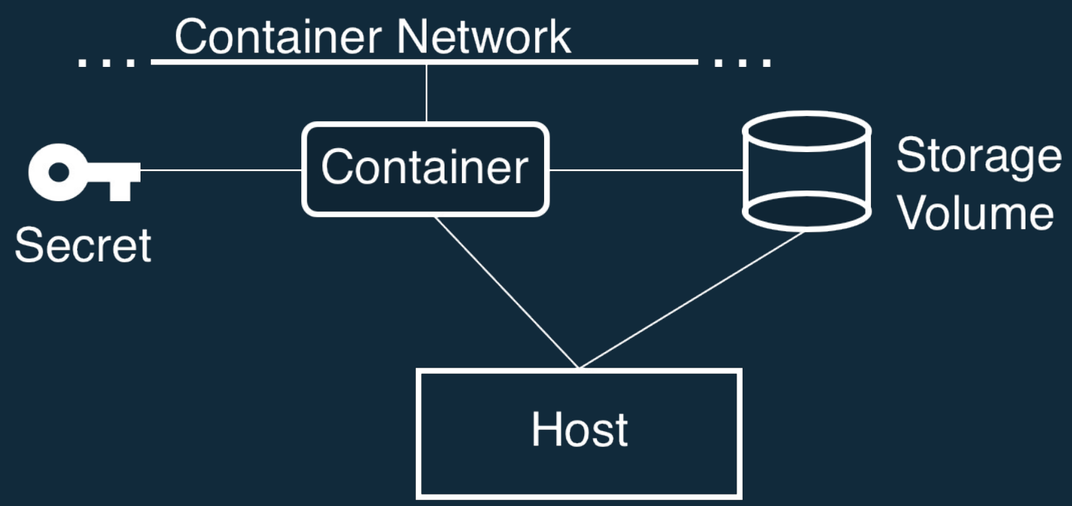
STACK RELATIONSHIP



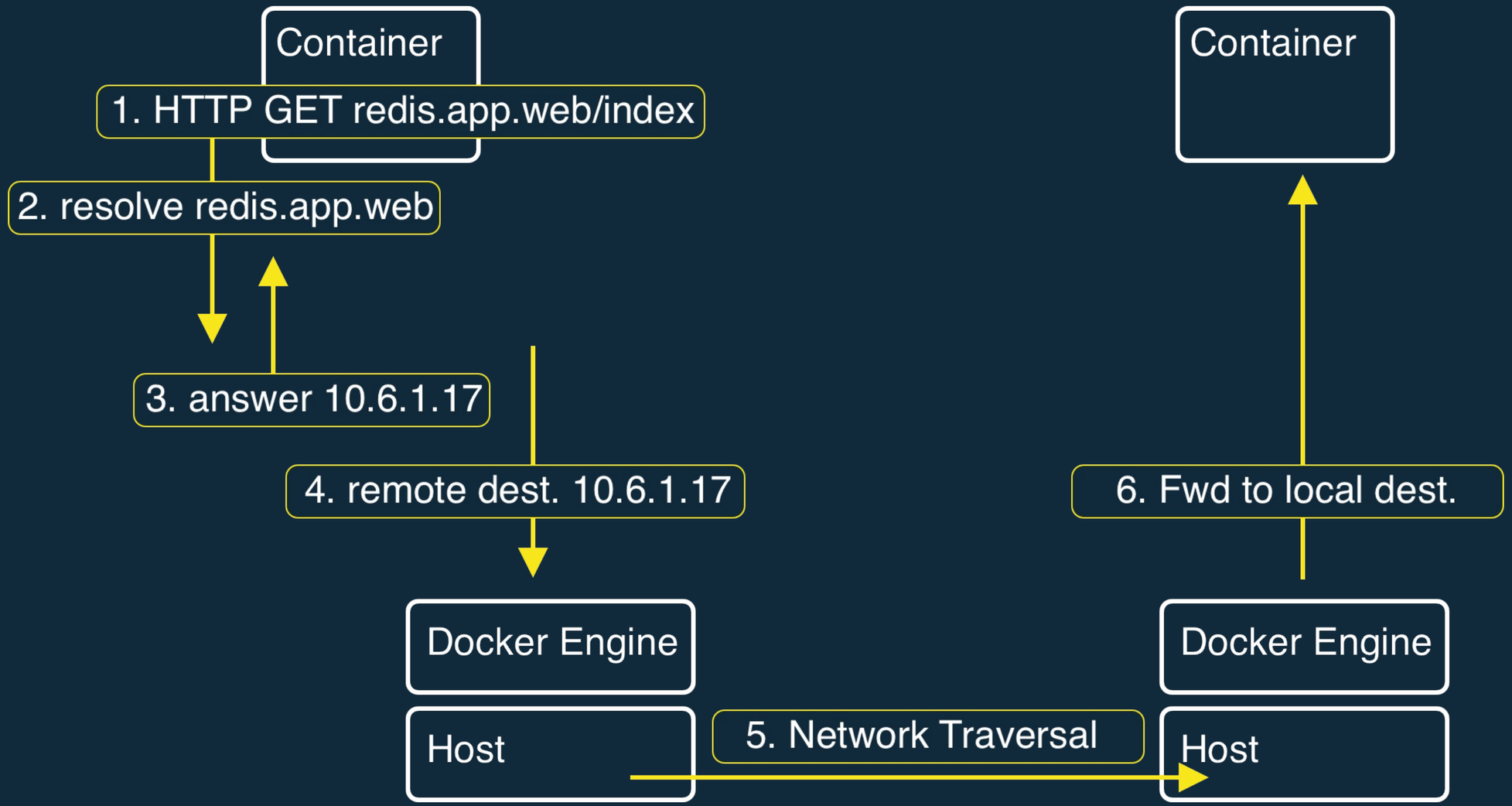
ONION & WEB RELATIONSHIPS

Onion

Web

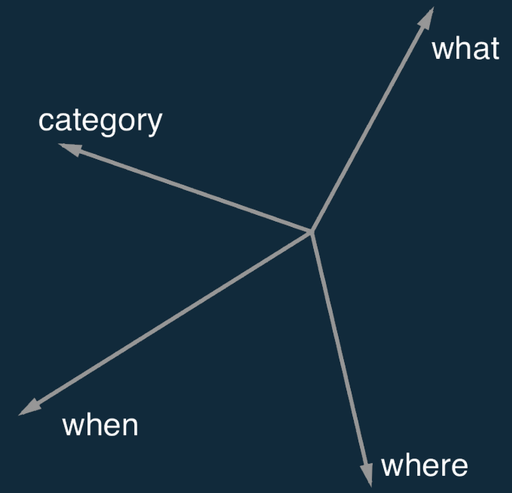


FLOW RELATIONSHIP

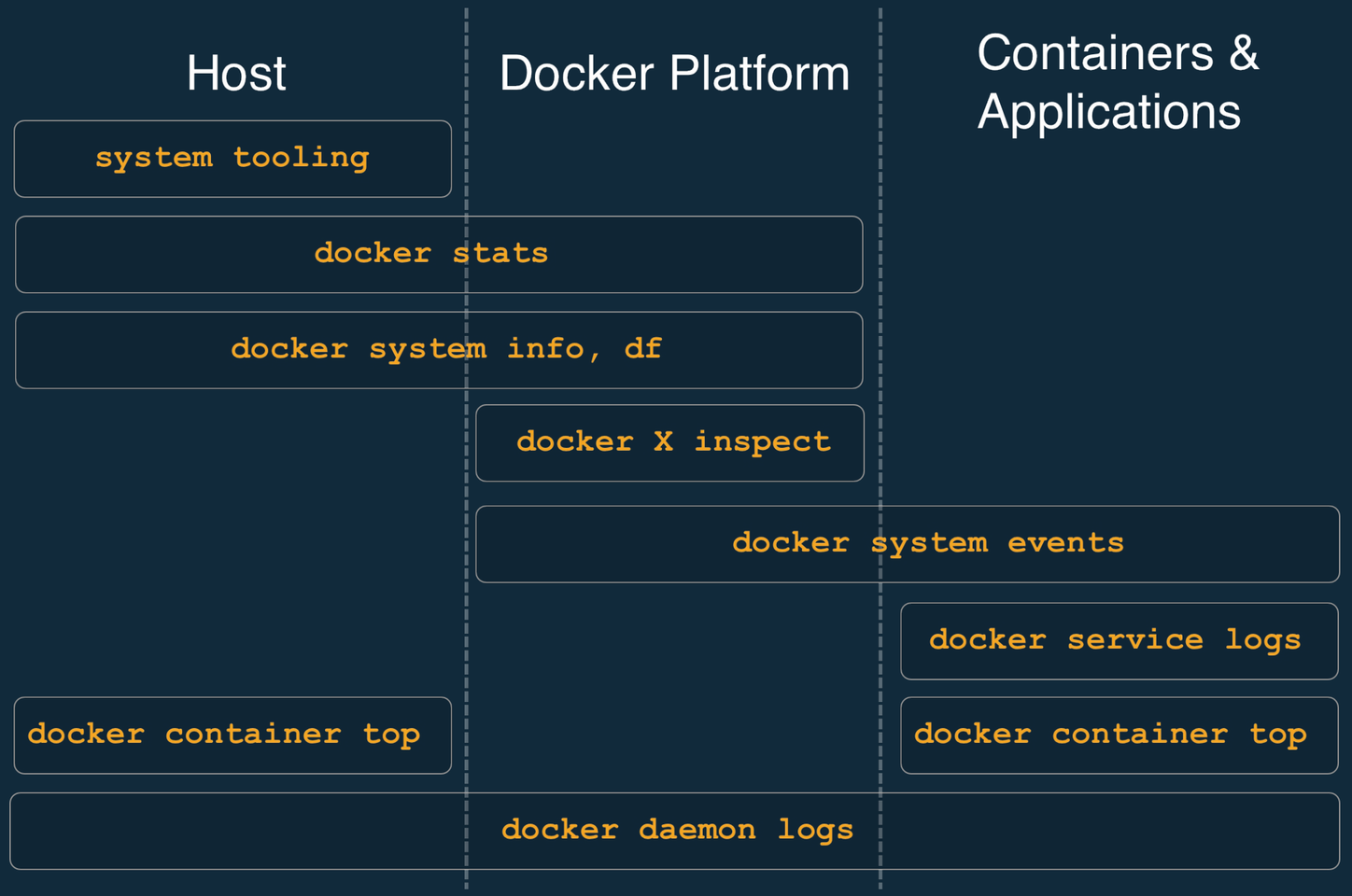


CONSTRAINING PROBLEMS

- Multi-dimensional problem space
- Dimensions: what, when, where, event category, etc.
- Cut down dimensions by process of elimination



SOURCES OF INFORMATION



SEVERITY TRIAGE

- P1 Production down: rollback immediately
- P2 Production negatively affected: brief analysis, rollback soon
- P3+ No production effect: prioritize analysis & develop solution

IDENTIFYING NON-PROBLEMS

- Individual containers are expendable
- When problems arise, kill it and schedule a new one
- Orchestrators can do this automatically
- This is expected behavior and not a problem.

VMs

Containers

Dog photo jeffreyw; Livestock photo Paul Asman, Jill Lenoble; images CC-BY 2.0

EXERCISE: PROBLEM SOLVING STRATEGIES

Work through:

- Tracing Relationships
- Narrowing Down Issues

in the Docker Troubleshooting & Support Exercises book.

FURTHER READING

- Troubleshooting 201: Ask the Right Questions: http://bit.ly/2zSu5eO
- Swarm Troubleshooting Methodology: https://dockr.ly/2JgM6VK

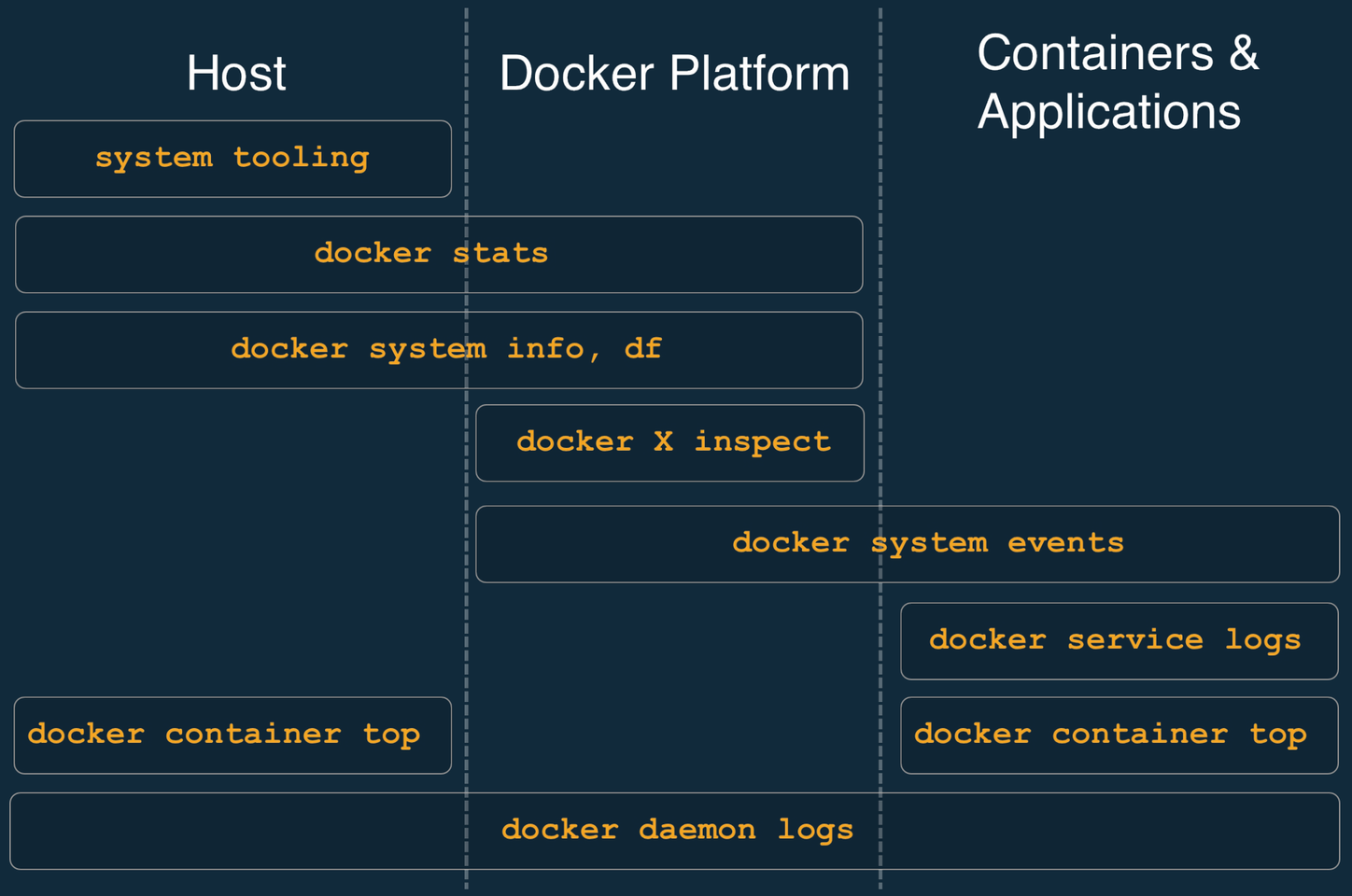
LOGGING & MONITORING STRATEGIES

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Identify all the sources of information provided by the Docker platform
- Configure logging outputs for convenient ingestion in a logging platform
- Identify some common elements of the Docker platform to build alerts around

SOURCES OF INFORMATION



MINING LOGS

- Timestamps correlate otherwise hard-to-relate objects
- JSON parsable by anything with a REST API

TIMESTAMPS & JSON

- Output as JSON with `--format '{{json .}}'`; works with:
  - `docker stats`
  - `docker system info`
  - `docker system events`
- `docker [X] inspect [Y]`: JSON by default
- Only `system events` automatically timestamps events; make sure to collect timestamp for all others on call.

JOURNALD LOGS

- Linux's per-node log aggregator
- Assembles docker daemon logs with kernel and other host info
- Can configure container logs to get sent to the journal
- Timestamped by default, format as JSON with flag

DOCKER-SPECIFIC ALERTS

- OOMKilled in container inspect
- Duplicate IPs
- Container healthcheck failures
- Network latency metrics (can damage management consensus)
- Scheduling failures
- NOT isolated container failures

EXERCISE: MONITORING ENDPOINTS

Work through:

- Monitoring Endpoints
- Running Docker Daemon in Debug Mode

in the Docker Support & Troubleshooting Exercise book.

FURTHER READING

- --format flag syntax: https://dockr.ly/2LChSOx
- A simple, containerized network monitor demo: https://bit.ly/2kxzChO

DOCKER DOCUMENTATION

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Find relevant documentation provided by Docker to solve a specific issue
- List at least 3 sources of technical documentation provided by Docker

DOCKER PRODUCT DOCUMENTATION

- docs.docker.com
- Starting point for simpler "how do I use Feature X?" questions
- /ee: features and high-level structure
- /reference: CLI, API docs

DOCKER REFERENCE ARCHITECTURES

- success.docker.com/architectures
- Written by Docker field engineers
- Deep dives appropriate for "what's the best way to do X?" questions
- Homework: read all of these!

DOCKER SOLUTIONS

- success.docker.com/solutions
- Written by Docker support & field engineers
- Deep dives appropriate for "how do I fix X?" questions
- Homework: read all of these!

EXERCISE: DOCUMENTATION

Work through:

- Documentation

in your exercise book.

in the Docker Troubleshooting & Support Exercises book.

FURTHER READING (1/2)

- Docker Documentation: https://docs.docker.com/
- Docker Success Center: https://success.docker.com/
- Docker Support Site: https://support.docker.com/
- Docker Training: https://success.docker.com/training
- Play with Docker Classroom: https://training.play-with-docker.com/

FURTHER READING (2/2)

THE MOBY PROJECT DOCUMENTATION

- Moby - Toolkit for SW Containerization
  - Docs: http://mobyproject.org/
  - GitHub: /moby/moby
- containerd - Container Runtime
  - Docs: https://containerd.io/
  - GitHub: /containerd/containerd
- Notary - Content Trust
  - Docs: https://docs.docker.com/notary/
  - GitHub: /theupdateframework/notary

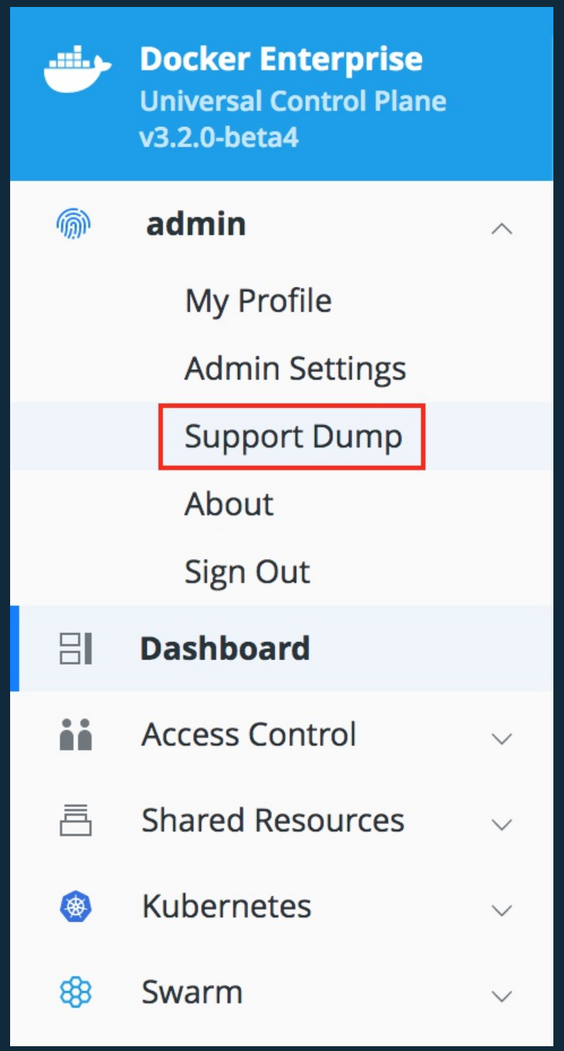
UCP SUPPORT DUMP

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Create a support dump, per cluster and per node
- Find system, orchestrator, and application level information in the UCP support dump
- Configure the Docker engine of a UCP node to run in debug mode without causing down time

GET A SUPPORT DUMP



- Linux single node dump:

```
docker run --rm \
    --name ucp --log-driver none \
    -v /var/run/docker.sock:/var/run/docker.sock \
    docker/ucp:version \
    support > docker-support-HOSTNAME.tgz
```

- Windows single node dump:

```
PS> docker container run --name windowssupport \
    -v 'C:\ProgramData\docker\daemoncerts:C:\ProgramData\docker\daemoncerts' \
    -v 'C:\Windows\system32\winevt\logs:C:\eventlogs:ro' \
    docker/ucp-dsinfo-win:version
PS> docker cp windowssupport:'C:\dsinfo' .
PS> docker rm -f windowssupport
```

SUPPORT DUMP: DSINFO

<hostname>/dsinfo/dsinfo.txt

1. Docker information:
   - docker [version | system info | image ls | stats]
   - daemon.json content
2. DTR Stats:
   - dtr-ol status
   - ulimits in each DTR container
3. System level stats:
   - CPU, memory, iops, network, iptables
4. Container stats:
   - container metadata
   - network info from inside network ns
5. Note IP overlap summary at bottom

SUPPORT DUMP: PER NODE INFO

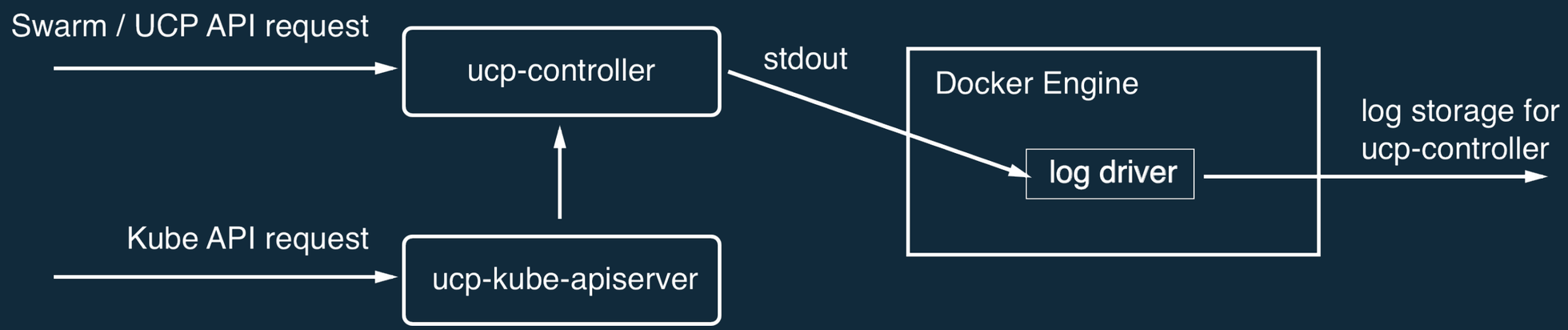
- Containers
  - inspect/: **docker container inspect** for all
  - logs/: **docker container logs** for all
- Docker Daemon
  - daemon-stack-trace.log
  - journalctl_daemon.log
- Certs
  - certdump.txt: ca, cert info from swarm and all system volumes

SUPPORT DUMP: TOP LEVEL

- UCP Agent info
- Kube describe all, nodes
- Interlock / HRM info
- Per-node folders

AUDIT LOGS

- Timestamp & user IDs invoking security-relevant UCP API calls
- Always available in container logs of ucp-controller
- Optionally available in support dump



EXERCISE: UCP SUPPORT DUMP

Work through:

- Installing UCP
- UCP Support Dump
- Manually Collecting a Support Dump

in the Docker Troubleshooting & Support Exercises book.

FURTHER READING

- Get Support: https://dockr.ly/2Jm9AIy
- How do I create a Support Dump?: http://dockr.ly/2AGSssB
- docker/ucp support: https://dockr.ly/2JjlEsQ

TROUBLESHOOTING RESOURCE PROBLEMS

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Generate a report of the memory usage on a UCP node
- Analyze I/O bottlenecks on a UCP node
- Identify the source of high CPU load on a UCP node

COMMON PROBLEMS

- Out Of Memory (OOM)
- I/O Bottlenecks
- CPU Overload

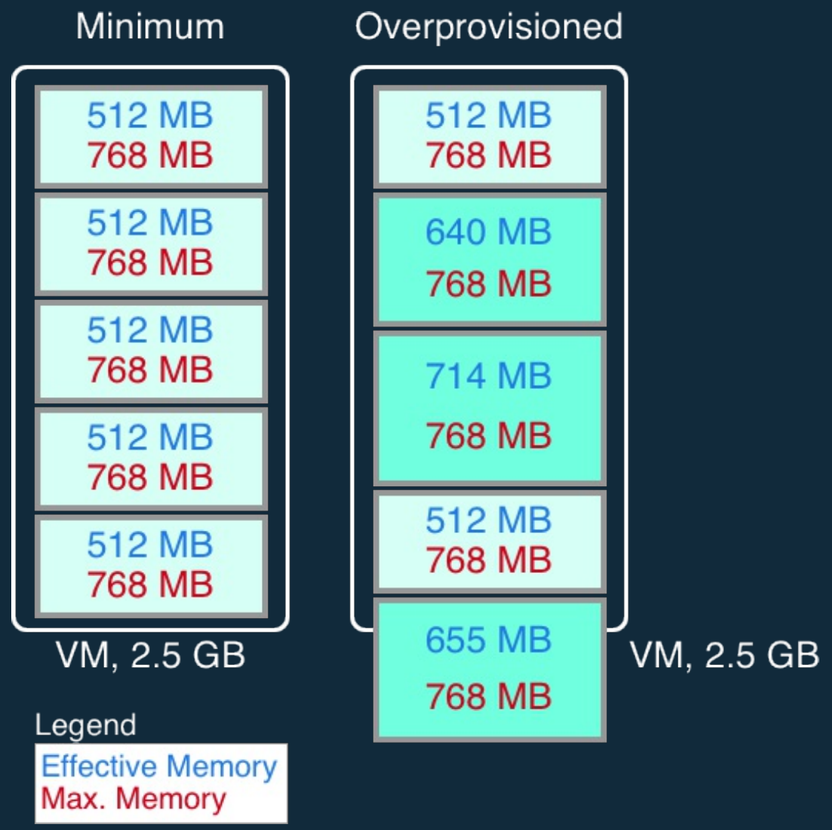
OUT OF MEMORY

Problem:

- Linux kills random process when OOM
- Can be Docker daemon!
- Or, total system blockage

Solution:

- Always limit memory
- Never overprovision



I/O BOTTLENECKS

Problems:

- R/W Container Layer
- Storage with high Latency
- High Concurrency
- Low Bandwidth to DB
- Chatty application

Possible Solutions:

- Volume drivers with low latency
- Reduce chattyness of application
- Use queueing system
- Increase bandwidth
- Batch requests

CPU OVERLOAD

Problem:

- Noisy Neighbor
- Too many Processes

Solution:

- Limit CPU access

WHAT DOES DOCKER OFFER?

cgroup support via:

What	Applies to	Flags
Memory	Service	--limit-memory & --reserve-memory
CPU	Service	--limit-cpu & --reserve-cpu
I/O	Container	--device-[read|write]-bps, --device-[read|write]-iops

EXERCISE: TROUBLESHOOTING RESOURCE PROBLEMS

Work through:

- Troubleshooting Resource Problems
- Inspecting Namespaces

in the Docker Troubleshooting & Support Exercises book.

FURTHER READING

- Limit a Container's Resources: http://dockr.ly/2wqN5Nn
- Linux OOM Killer: https://linux-mm.org/OOM_Killer
- OOM Management: http://bit.ly/2jQH1vC
- What is Overcommit, and why is it bad: http://bit.ly/2jfPJj4
- Docker Reference Architecture: Running Docker Enterprise Edition at Scale: http://dockr.ly/2DW9R3n

TROUBLESHOOTING NETWORK PROBLEMS

LEARNING OBJECTIVES

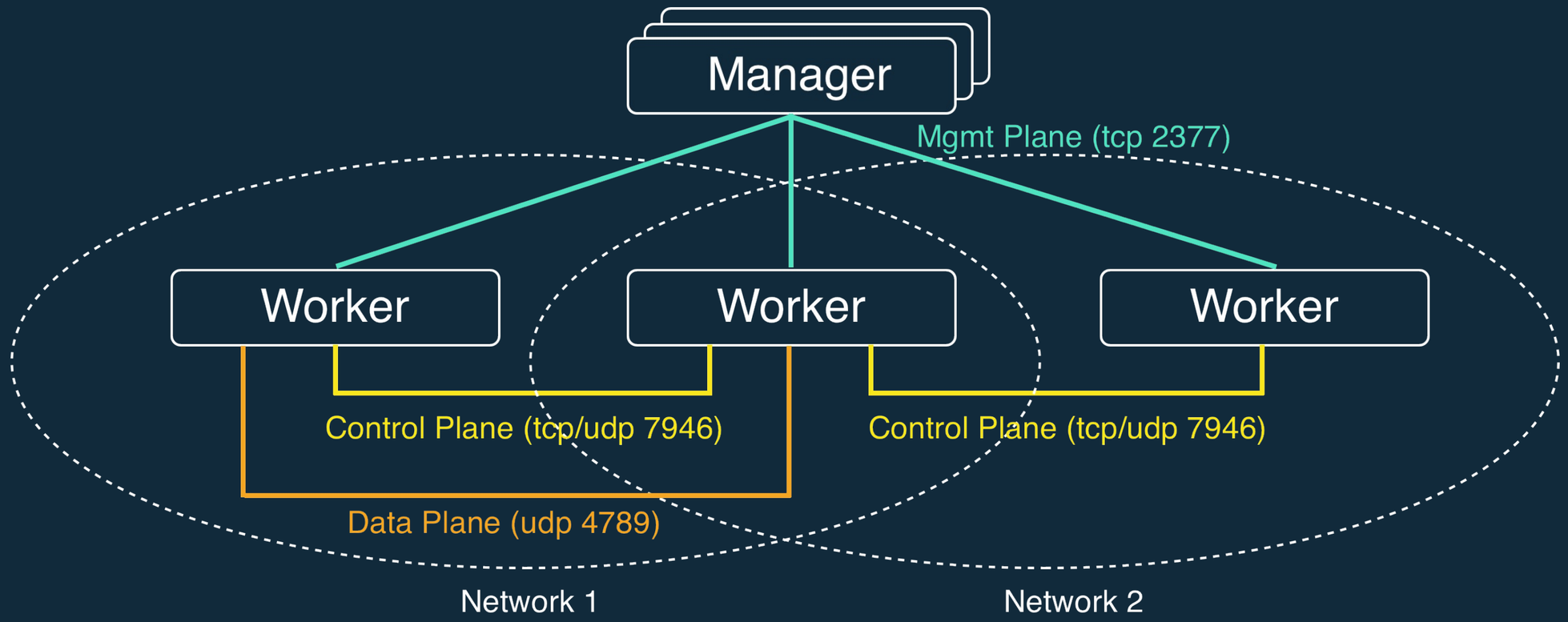
By the end of this module, learners will be able to:

- Identify management, control and data plane components for Swarm and Kube with Calico
- Test connectivity and service discovery in Swarm and Calico
- Identify and test connectivity between UCP and DTR components

ORCHESTRATOR NETWORKING PLANES

- Management: raft consensuses, scheduling, EE infra
- Control: DNS maintenance, service discovery
- Data: container to container communication

SWARM NETWORKING



SWARM PATHOLOGIES

- Can't join swarm, can't schedule workload:

1. check tcp/2377

- Can't resolve container name:

1. retry connection
2. check Docker network connection
3. check udp+tcp/7946

- Successful name resolution but failed connection:

1. check Docker network connection
2. check udp/4789

Note if data or control plane ports were blocked at time of node join, you will likely need to remove and rejoin the node after unblocking these ports.

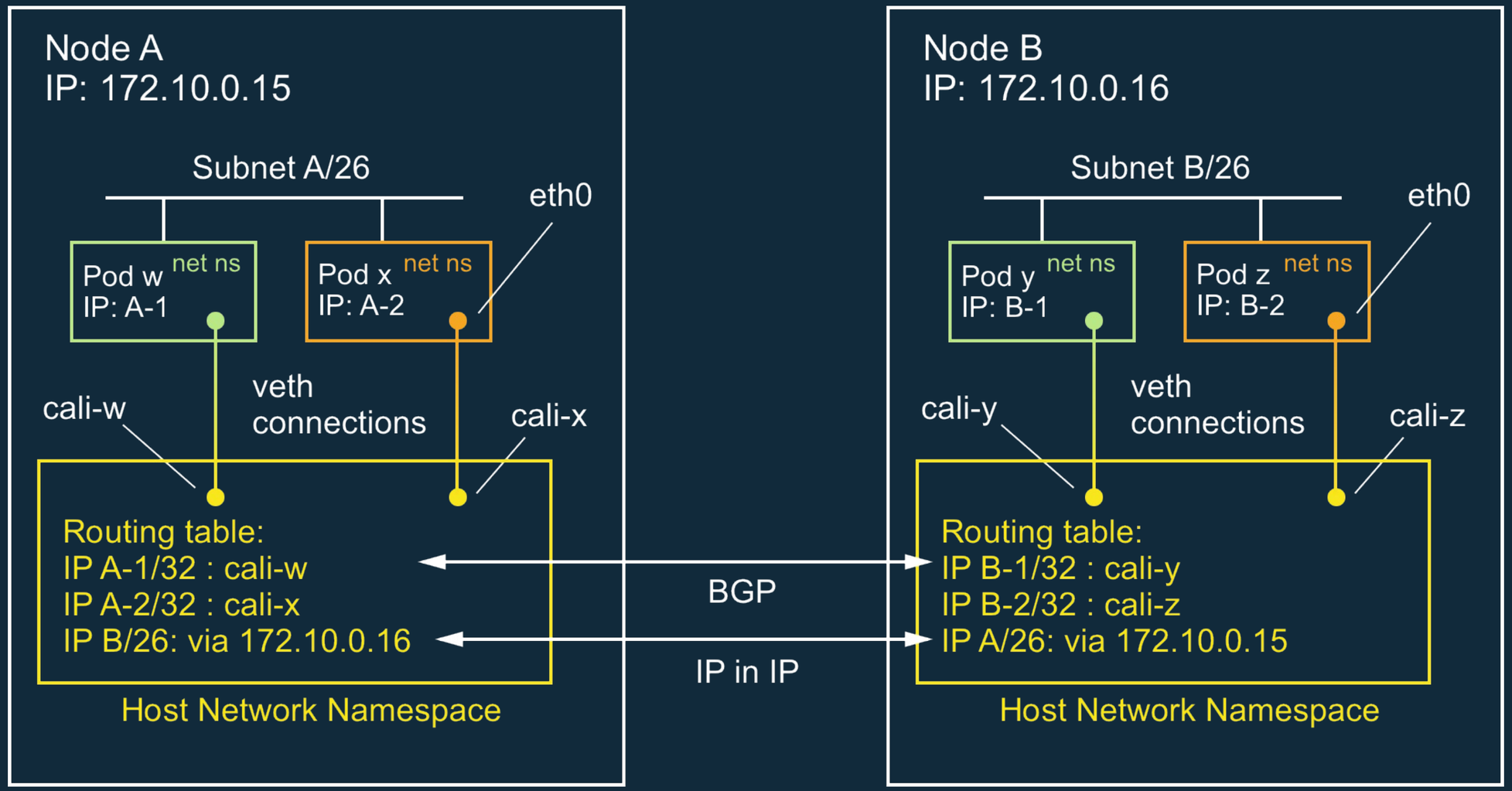
UCP NETWORKING REQUIREMENTS

- Same topology as swarm planes
- Ports 12xxx: UCP inter-component communication
- 179, 6443, 6444, 10250: Kube API, Kubelet, networking
- Complete topology:
  - List: https://dockr.ly/2yBGmlV
  - Figure: See 'UCP Architecture' appendix in your exercise book

UCP PATHOLOGIES

- Unhealthy etcd: manager to manager tcp/12380 (etcd raft)
- Unhealthy rethink: manager to manager tcp/12384 (rethink raft)
- 'Classic node inventory': node to manager tcp/12379 (etcd api)
- Metrics: manager to node tcp/12376
- Kube management plane: manager to worker tcp/10250; worker to manager tcp/6443

CALICO CONTROL & DATA PLANES



SERVICE DISCOVERY & ROUTING

- Service discovery: DNS lookup
- Swarm routing: VIP resolution
- Kube routing: clusterIP and nodePort services

SERVICE DISCOVERY: DOCKER DNS

- DNS lookup by container or service name
- Scoped to software defined networks
- Typical /etc/resolv.conf:

```
search ec2.internal
nameserver 127.0.0.11
options ndots:0
```

- Don't forget about eventual consistency

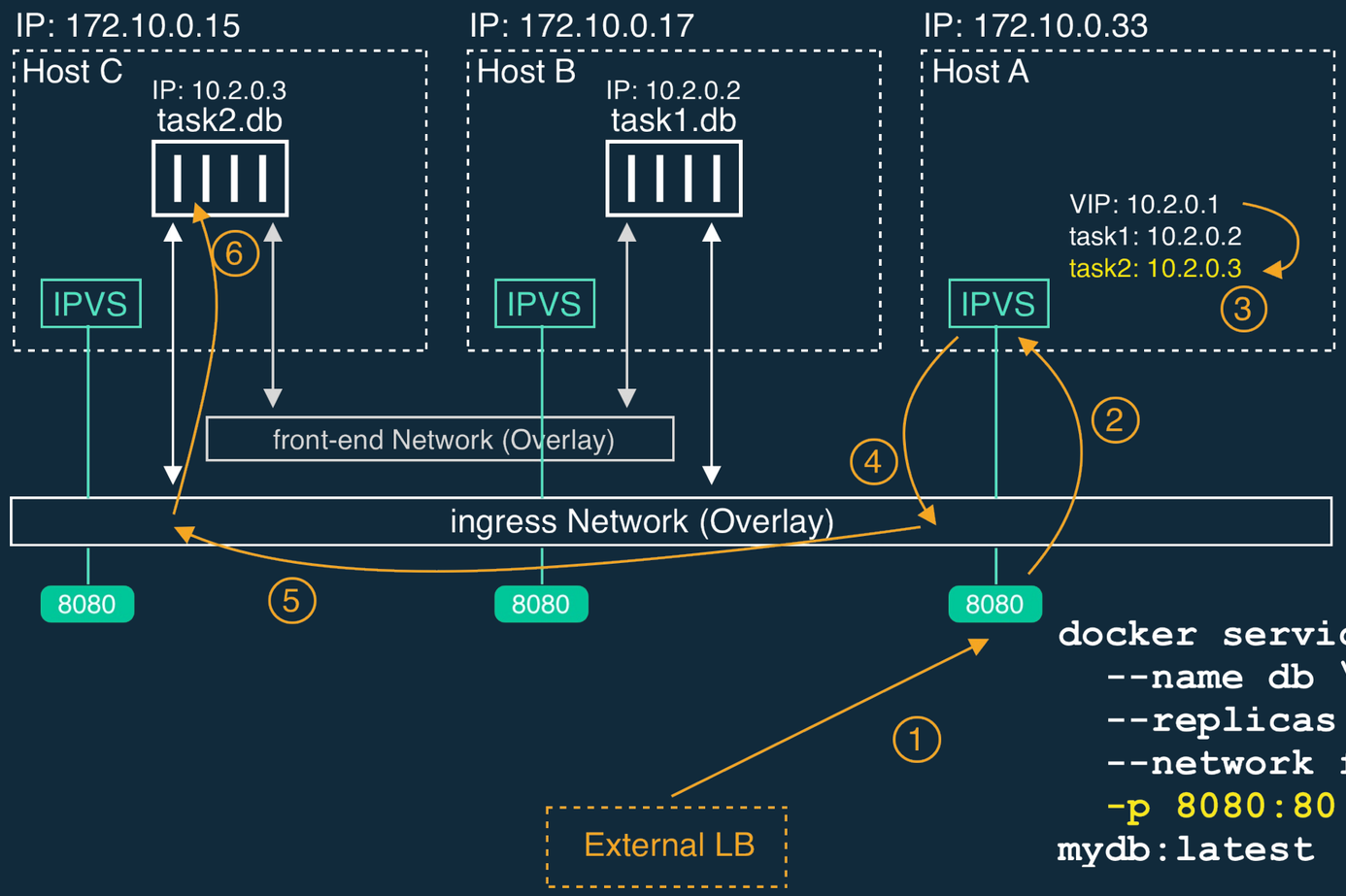
SERVICE DISCOVERY: KUBE DNS

- Must be installed on top of Kube
- DNS lookup by kube service and namespace
- Typical /etc/resolv.conf:

```
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local ec2.internal
options ndots:5
```

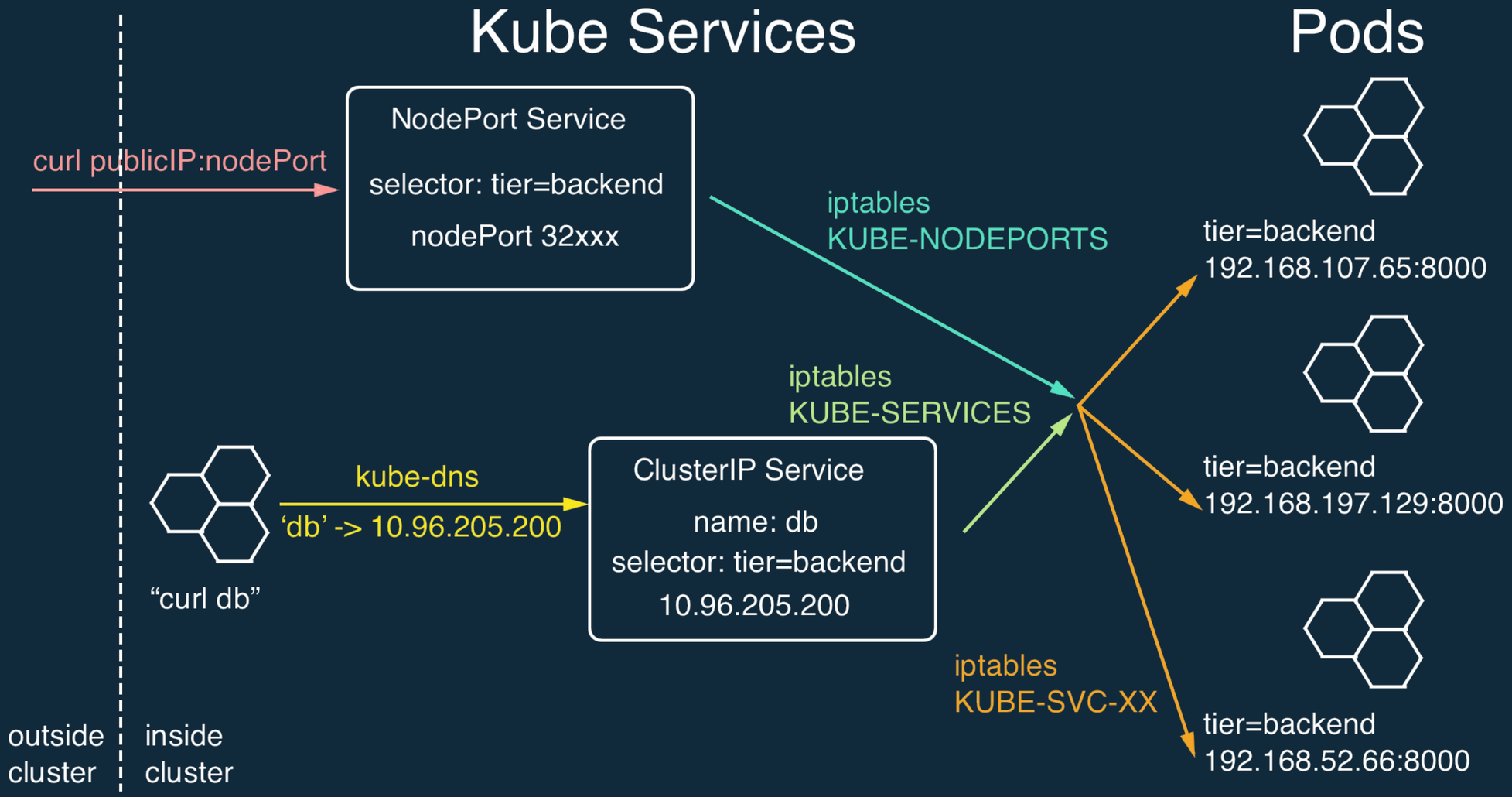
- nameserver == clusterIP of kube DNS service

SWARM VIP RESOLUTION




```
docker service create \
  --name db \
  --replicas 2 \
  --network front-end \
  -p 8080:80 \
mydb:latest
```

KUBE SERVICE ROUTING



HIGH LATENCY

- Unhealthy [rethink | etcd | manager]: round trip time > 50 ms
- Consider VXLAN bandwidth overhead (50 bytes per frame)
- Consider frequency of container communications ('chattyness')

TOOLING

- Connectivity: nmap, netcat
- Latency: iperf, netcat
- DNS resolution: drill
- VIP resolution: nslookup

all these and more available in the nicolaka/netshoot container

EXERCISE: TROUBLESHOOTING NETWORK PROBLEMS

Work through

- Troubleshooting Network Problems
- Troubleshooting Swarm Networking Planes
- Troubleshooting Kubernetes Services

in the Docker Troubleshooting & Support Exercises book.

FURTHER READING

- Troubleshooting container networking: https://dockr.ly/2KFFu7j
- Troubleshooting Calico: https://bit.ly/2tTfaNl
- netshoot container: http://bit.ly/2z2xGn4
- Docker container networking: http://dockr.ly/1QnT6y8
- Understand container communication: http://dockr.ly/2iSrHO0

TROUBLESHOOTING UCP

LEARNING OBJECTIVES

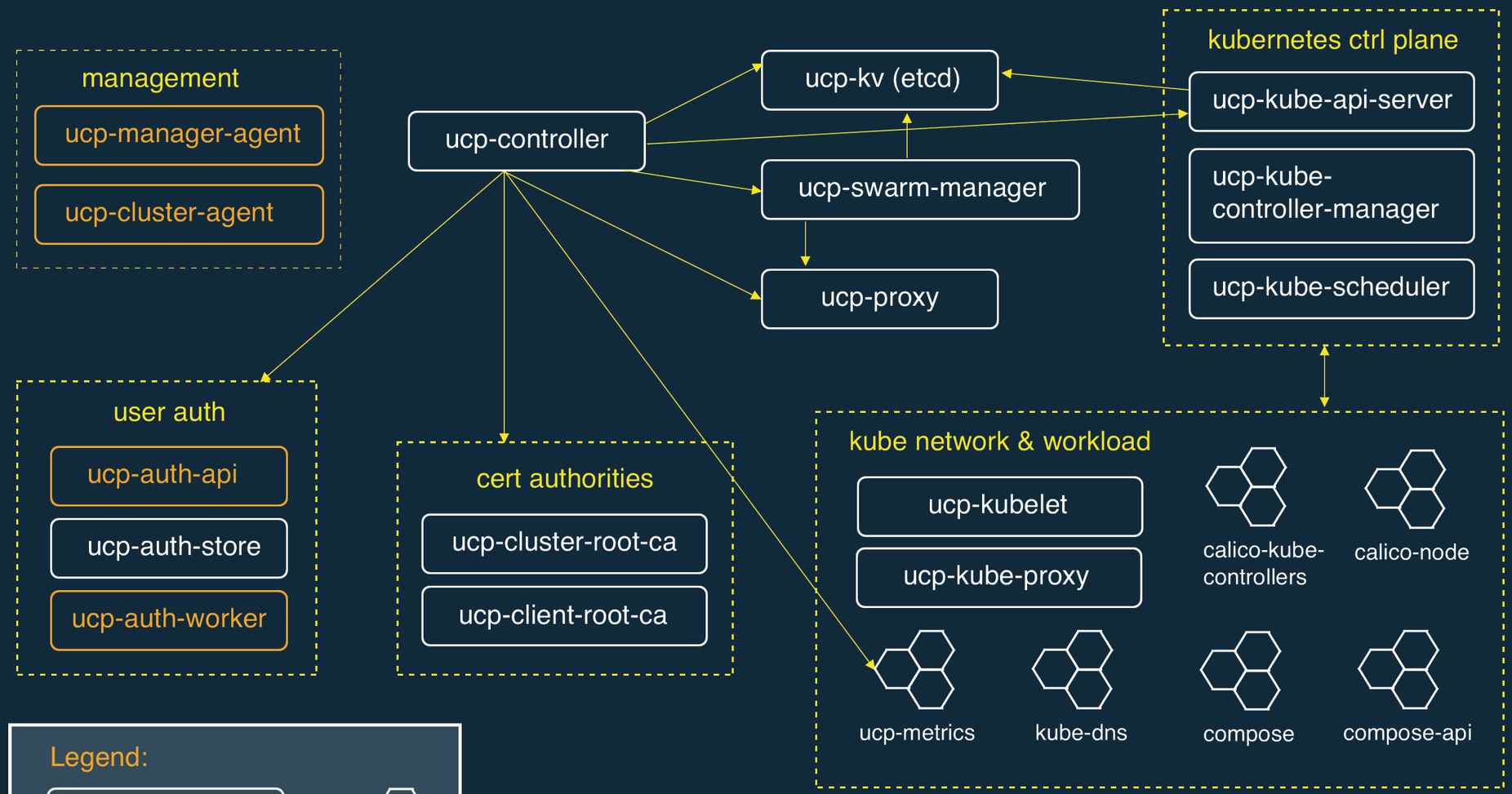
By the end of this module, learners will be able to:

- Install a highly available UCP swarm
- Describe the networking and relationships between UCP containers
- Identify likely sources of logs in common UCP malfunctions
- Repair broken etcd and rethinkdb databases

UCP REVIEW

- Provides RBAC, API, various authentication methods
- Containerized app running on a swarm
- Common troubleshooting pitfall: blaming UCP for Swarm problems, or vice versa

UCP CONTAINERS



UCP NETWORKING REQUIREMENTS

- Same topology as swarm planes
- Ports 12xxx: UCP inter-component communication
- 179, 6443, 6444, 10250: Kube API, Kubelet, networking
- Complete topology:
  - List: https://dockr.ly/2yBGmlV
  - Figure: See 'UCP Architecture' appendix in your exercise book

UCP LOGGING SOURCES

	ucp-controller	swarm manager	auth-api	auth-store/ rethinkdb	auth-worker	ucp-kv/etcd	ucp-metrics	ucp-agent/ reconcile	Engine
Volumes, Containers, Networks	●	●		○		○			●
Services, Secrets, Configs	●		○	○	●	○			●
Authentication Failure	○		●	●	○			●	
Authorization Failure	●		○	●		●		○	
Node list timeout / error	●	●						○	●
Unhealthy nodes	●	●	○					●	●
Node stuck [Pending]		○						●	
“rpc error …”									●
“context deadline exceeded”	●							○	●
Metrics not loading	●						●	○	●

● Strong candidate
○ Weak candidate

STATE IN UCP

- Most fragile state persisted in etcd and rethinkdb
- Both replicate across managers as rafts
- Replication process can be corrupted (primarily) by network problems
- All state pushed out to volumes

UCP SURGERY

- com.docker.ucpagent-pause=true label suspends UCP reconciliation
- etcdctl packaged with ucp-kv
- docker/ucp-auth image has several flags for manipulating rethinkdb
- Containerized rethinkdb clients available

EXERCISE: TROUBLESHOOTING UCP

Work through:

- Troubleshooting UCP etcd and RethinkDB

in your exercise book.

in the Docker Troubleshooting & Support Exercises book.

FURTHER READING

- High-level UCP troubleshooting: https://dockr.ly/2shJsZh
- Troubleshooting UCP databases: https://dockr.ly/2LDMxuK
- Understanding UCP Node State Messages: https://dockr.ly/2yAxVdi

TROUBLESHOOTING DTR

LEARNING OBJECTIVES

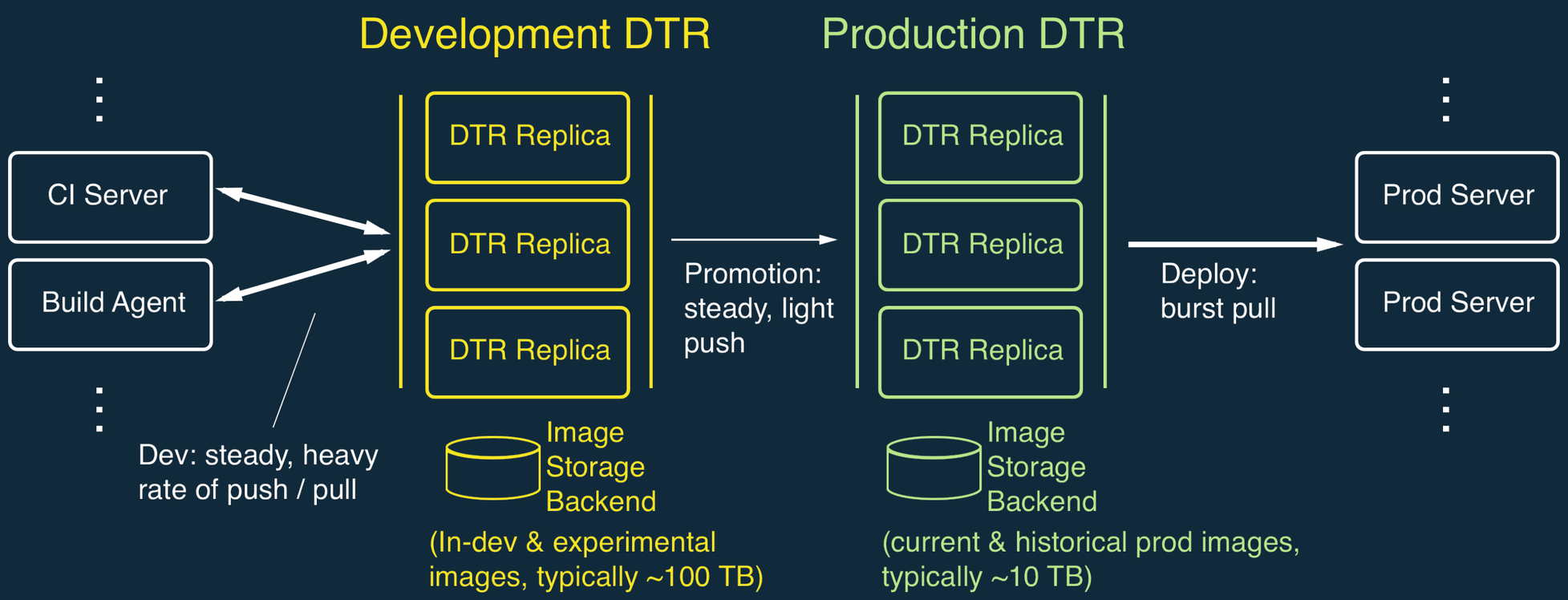
By the end of this module, learners will be able to:

- Install a highly available DTR cluster
- Attribute DTR performance to deployment decisions
- Localize DTR networking problems
- Repair a DTR cluster from a single healthy replica

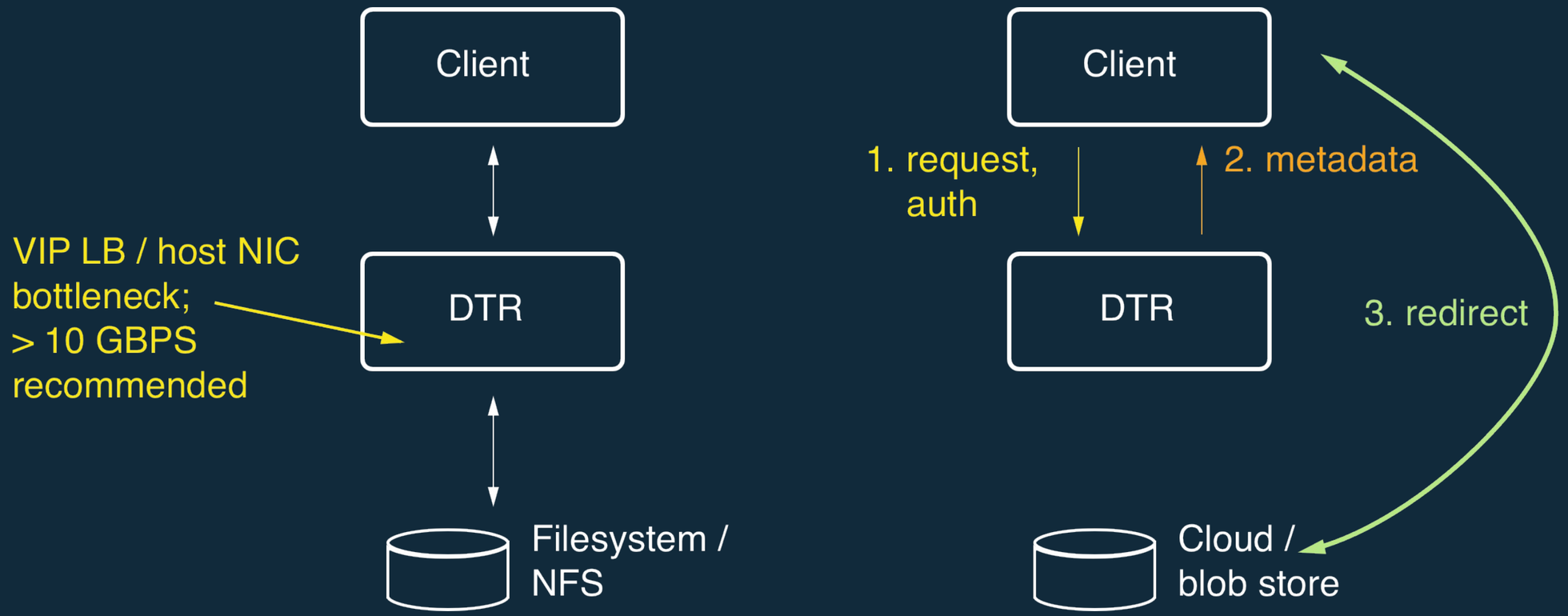
DTR REVIEW

- Containerized registry for managing images
- Runs on top of UCP & Swarm
- Relies on independent storage backing for holding images
- Provides security tooling like scanning and content trust
- Most common sources of problems: system resources and consensus maintenance

DTR USAGE PROFILES



TRANSACTION PERFORMANCE



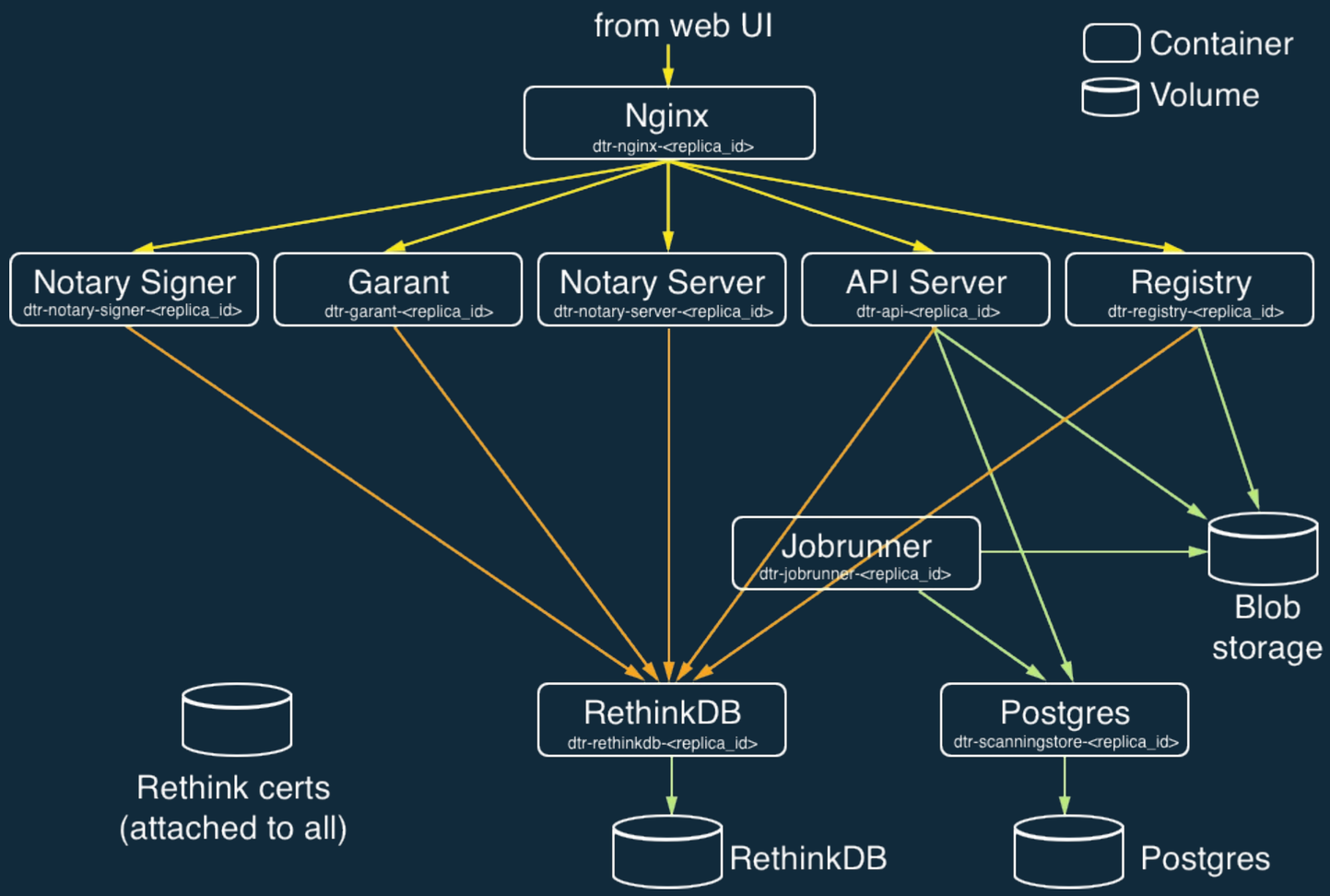
RESOURCE INTENSIVE OPERATIONS

- CPU: garbage collection, scanning
- memory: backup, replica join

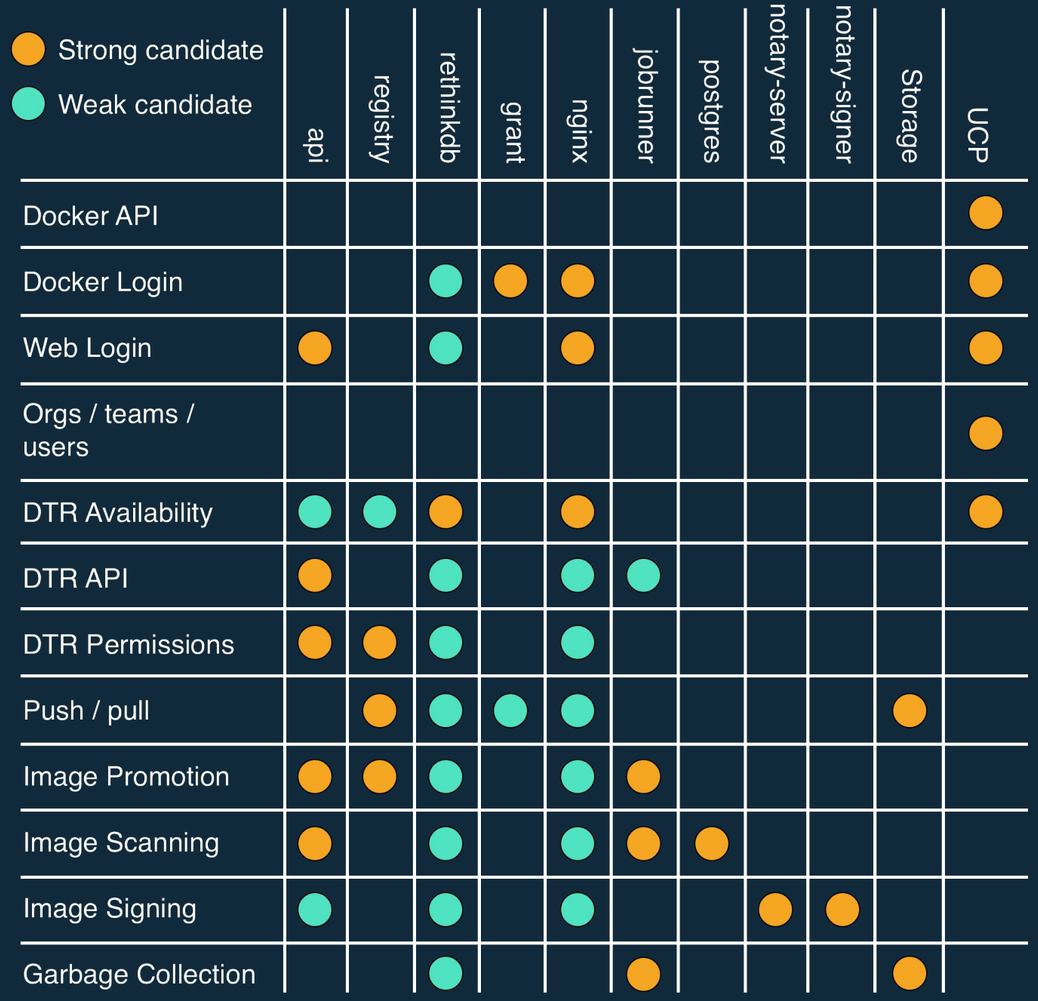
DTR RESOURCING SUMMARY

- Write uptime: limit garbarge collection
- Burst pulls: 10 GBPS NICs + LB or cloud / blob backing
- Scanning, GC, replica joins, backups: 4+ vCPU, >16 GB RAM
- Storage backing: ~10-~100 TB

DTR MICROSERVICES



DTR LOGGING SOURCES



AUDITING DTR ACTIONS

- Per repo: Activity monitor (ex: push, delete, scan, promote)
- All DTR: Job log (ex: scanning, garbage collection, webhooks, pruning)
- Available through DTR web frontend & API endpoints

REPLICA FAILURE TROUBLESHOOTING

Problems:

1. Inter-replica latency
2. Unhealthy rethinkdb replica(s)
3. Container loss

Solutions:

1. Measure latency on dtr-ol overaly
2. Fix RethinkDB from a healthy replica
3. Tear down and rejoin replica (last resort)

TROUBLESHOOTING THE NETWORK

On DTR node run:

```
docker container run -it --rm \
    --net dtr-ol --name overlay-test1 \
    --entrypoint sh docker/dtr
```

On different DTR node run:

```
docker container run -it --rm \
    --net dtr-ol --name overlay-test2 \
    --entrypoint ping docker/dtr -c 3 overlay-test1
```

TROUBLESHOOT NOTARY

Get Audit Log from

```
https://<dtr-url>
```

- All Repos:

```
GET /v2/_trust/changefeed
```

- Individual Repo:

```
GET /v2/<dtr-url>/<repository>/_trust/changefeed
```

- Admin sees all
- Users only see repos they have read access

Sample Log Entry:

```
{
   "count": 1,
   "records": [
      {
         "ID": "0a60ec31-d2aa-4565-9b74-4171a5083bef",
         "CreatedAt": "2017-11-06T18:45:58.428Z",
         "GUN": "dtr.example.org/library/wordpress",
         "Version": 1,
         "SHA256": "a4ffcae03710ae61f6...",
         "Category": "update"
      }
   ]
}
```

EXERCISE: TROUBLESHOOTING DTR

Work through:

- Installing DTR
- Troubleshooting DTR

in the Docker Troubleshooting & Support Exercises book.

FURTHER READING

- Docker Trusted Registry: https://dockr.ly/2F84rkN
- DTR API Documentation: http://dockr.ly/2nzxNVE
- Notary: http://dockr.ly/2zS7E6q
- RethinkDB: https://www.rethinkdb.com/
- DTR Job logs: https://dockr.ly/2Dg4YFK

DISASTER RECOVERY

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Make a disaster recovery plan
- Back up Swarm, UCP and DTR
- Restore a cluster from backups

DISASTER STRIKES

- Quorum loss
- Production down
- General catastrophe

BACKUP & RESTORE ORDER

1. User volumes & images (if necessary)
2. Swarm
   - Captures service config, secrets, overlay networks, raft keys
3. UCP
   - Captures user data, UCP config & volumes
4. DTR
   - Captures DTR config & volumes (NOT images!)

SWARM BACKUPS

1. Stop Docker
2. Archive and encrypt /var/lib/docker/swarm
3. Restart Docker

- Does not interrupt workloads
- Make sure quorum is not lost during procedure
- Hot backups possible but not recommended

UCP BACKUPS

- Performed by `docker/ucp:<version> backup` command
- Archives all UCP volumes
- Does not interrupt UCP operation ('hot backup')
- Disconnects any processes attached to UCP containers (exec, logs, attach)
- Use the same bootstrapper version that installed UCP
- Encrypt backups to protect keys

DTR BACKUPS

- Performed by `docker/dtr:<version> backup` command
- Does not stop DTR replica
- Encrypt backups to protect keys

RESTORING SWARM

On one node:

1. Stop Docker
2. Remove the contents of the `/var/lib/docker/swarm`
3. Restore `/var/lib/docker/swarm` from backup
4. Restart Docker
5. Reinitialize Swarm with `docker swarm init --force-new-cluster`
6. Verify Swarm state
7. Rejoin fresh managers and workers

RESTORING UCP

1. Ensure Swarm is healthy
2. Uninstall UCP if present
3. Restore UCP from backup using **restore** command from UCP bootstrap image
4. Double check that all nodes report healthy in UCP

RESTORING DTR

1. Ensure Swarm and UCP are healthy
2. Use **destroy** command from DTR bootstrapper to remove any existing DTR data
3. Recreate one DTR replica from backup
4. Rejoin additional replicas for HA

EXERCISE: DISASTER RECOVERY

Work through:

- Disaster Recovery

in your exercise book.

in the Docker Troubleshooting & Support Exercises book.

FURTHER READING

- Backing up a Swarm: https://dockr.ly/2iQGQvW
- Back up and restore UCP: https://dockr.ly/2qY6Zgf
- Back up and restore DTR: https://dockr.ly/2xJg03q

ENGAGING DOCKER SUPPORT

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Write a comprehensive and clear description for a Docker support case
- Determine the priority of a given support issue
- Know what to expect from a Docker Support response

WHEN ALL ELSE FAILS

- Try all the strategies in this course
- Log in at https://support.docker.com
- Include in your ticket:
  - UCP Support Dump (always)
  - Describe what you did (for active failures)
  - Describe what the state / config of your system was (for passive failures)
  - Describe what happened (with logs)
  - Describe what you expected to happen
  - Describe your Docker use case

Case Priority	Definition
Urgent: P1	Full production outage involving Docker EE, UCP, and/or DTR
High: P2	High impact to production services or severe impact to non-critical business operations
Normal: P3	Moderate impact to business operations
Low: P4	Low or no impact to business operations

SERVICE LEVELS

	Business Day SLA	Business Critical SLA
P1	Initial: 2 business hours; Follow-on: 2 business hours	Initial: 1 hour; Follow-on: 2 hours
P2	Initial: 4 business hours; Follow-on: 4 business hours	
P3	Initial: 8 business hours; Follow-on: 16 business hours	
P4	16 business hours; Follow-on: 32 business hours	

FURTHER READING

- Docker Support Site User's Guide: https://dockr.ly/2sRfGtX
- What should I include when opening a support ticket?: https://dockr.ly/2Md3wUY
- Commercial Support Service Levels: https://dockr.ly/2l3BWxn

TROUBLESHOOTING & SUPPORT SIGNATURE ASSIGNMENT

TROUBLESHOOTING SCENARIOS

- Several scripted scenarios
- Based on your cluster w/ 3 UCP managers + 1 DTR replica
- Do not look at the scripts!
  - Just run them without opening them
  - If you're stuck for >30 min, then see the solution inside

SIGNATURE ASSIGNMENT: DIAGNOSE & REPAIR

Work through:

- Diagnose & Repair

in your exercise book.

in the Docker Troubleshooting & Support Exercises book.

BONUS TOPICS

TROUBLESHOOTING ENVIRONMENTS

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Create a local Docker Swarm as a troubleshooting environment with VirtualBox, Docker Machine and Vagrant

VIRTUAL BOX AND DOCKER MACHINE

- Docker Toolbox:
  - docker-machine
  - VirtualBox
-
```
docker-machine create --driver virtualbox demo
docker-machine ssh demo
```
- Default: spins up a boot2docker ISO with latest Docker CE

VAGRANT

- Specify arbitrary environments
- Infrastructure as code
- Cons: Setting up Vagrant can be hard

EXERCISE: TROUBLESHOOTING ENVIRONMENTS

Work through:

- Using VirtualBox
- Using Vagrant

In the Docker Troubleshooting & Support Exercises book.

TROUBLESHOOTING ENVIRONMENT TAKEAWAYS

- Support relies on reproducible environments
- VirtualBox: simple environments
- Vagrant: More complex environments

FURTHER READING

- Virtualbox: http://bit.ly/2riK1Uw
- Vagrant: http://bit.ly/2eBcZWD
- AWS Cloudformation getting started: http://amzn.to/2DP0nqL
- Azure Resource Manager: http://bit.ly/2j7KCAW
- Pythonic SDK for AWS: http://amzn.to/2rQ6yUr

DOCKER TROUBLESHOOTING & SUPPORT

Thanks for coming! Please take our feedback survey:

https://bit.ly/2Mesp1L

Get in touch: training@docker.com

success.docker.com/training

YOU'RE ON YOUR WAY TO BECOMING DOCKER CERTIFIED!

- Study up with our Study Guides at http://bit.ly/2yPzAdb
- Take it online 24 hours a day
- Results delivered immediately
- Benefits include:
  - Digital certificate
  - Online verification
  - Private LinkedIn group
  - Exclusive events

SUCCESS.DOCKER.COM/CERTIFICATION