



TROUBLESHOOTING & MONITORING ARTIFACTORY & XRAY

WHO AM I?



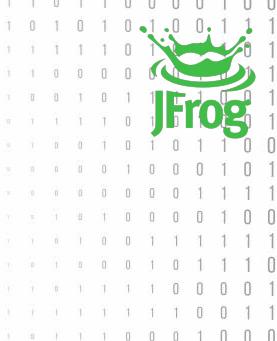
Jonathan ROQUELAURE

Senior Solution Architect - Squad Leader

jonathanr@jfrog.com

@roquelaurej

FROGS?



JFROG IN A NUTSHELL



2008
Founded



5,000
Customers



400
Employees



Clients include
70%
of the
Fortune 100



\$230M
Raised to date



All Hybrid
From OSS to Multicloud



6 Products
from Git to K8S



9 locations
7 countries

TECHNOLOGY LEADERSHIP



Forbes
CLOUD 100 LIST



Deloitte
Technology Fast
500
2018
Winners



SD Times
The 2018
100 Award

THE FROG PHILOSOPHY



END-TO-END
PLATFORM

SCALES TO
INFINITY

RADICALLY
UNIVERSAL

CONTINUOUS
SECURITY

HYBRID AND
MULTI-CLOUD

INTEGRATED
ECOSYSTEM



HONORED TO LEAD

Internet & Software



Technology & Electronics



Banking & Finance



Engineering & Aerospace



Retail & Consumer



Education & Research





JFrog ENTERPRISE+



MISSION CONTROL & INSIGHT

Analyze and measure the flow



XRAY

VCS & CI

ARTIFACTORY

Clear security and compliance issues

Code & Build

Store and manage your binaries globally

DISTRIBUTION

Distribute to production site

Deploy to production



ACCESS

Manage authentication and authorization globally



ARTIFACTORY
EDGE



ARTIFACTORY
EDGE



ARTIFACTORY
EDGE

AGENDA

- Introductions
- Setup Environment
- Artifactory Internals
- *Artifactory Logging*
- *High HTTP Request Throuput*
- *Database Connections Exceeded*
- *Help, I've Got 1 GB Disk Left*
- If time permits
 - *JVM Memory Issues*
 - *Monitoring REST API*



INTRODUCE YOURSELF

Short introduction - Name, Company, Role



SOME BACKGROUND lecture time

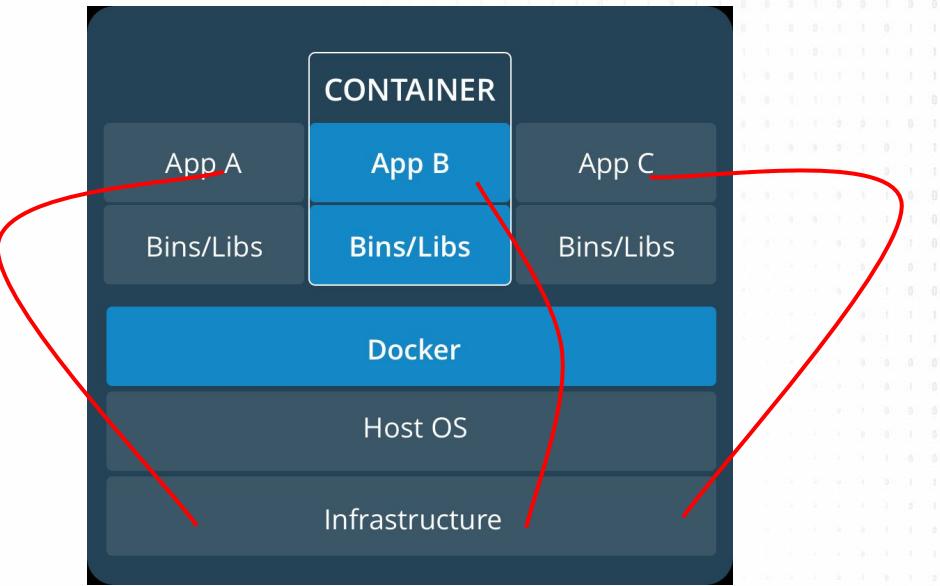


Let's talk about containers

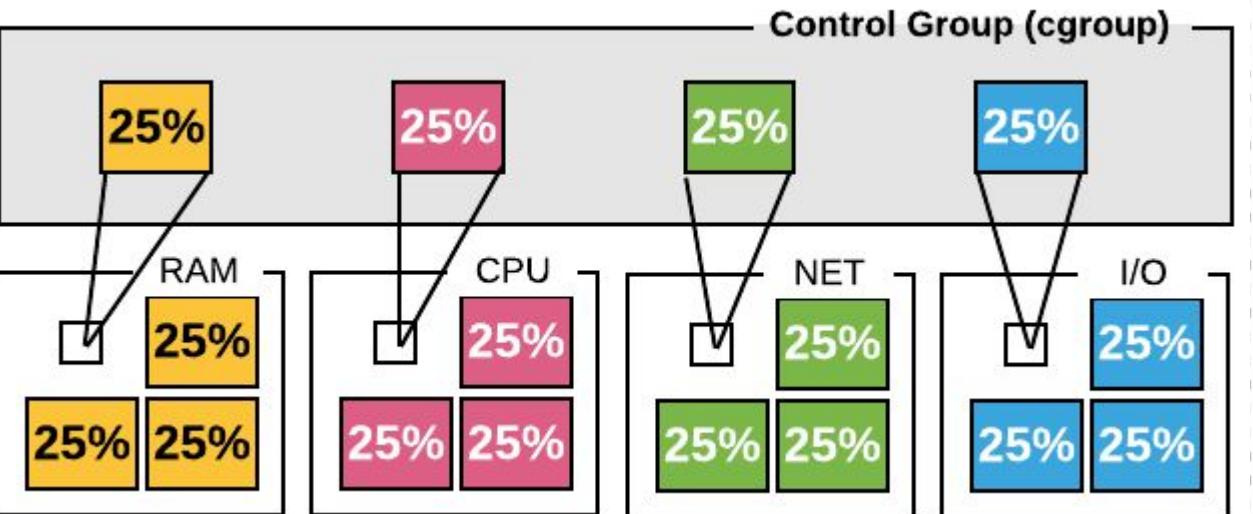


What is a container?

way to pack code and dependencies together
can run anywhere
execute multiple containers on a physical machine



cgroup - A Game Changer for Containers



Linux Control Groups and Process Isolation

Resource limitation: a group can be configured not to exceed a specified memory limit or use more than the desired amount of processors

Prioritization: one or more groups may be configured to utilize fewer or more CPUs or disk I/O throughput

Accounting: a group's resource usage is monitored and measured

Control: groups of processes can be frozen or stopped and restarted

Sounds familiar?

same concept as virtual machines

pack OS and software together, to run in isolated instances

can run anywhere the specific hypervisor runs

multiple VMs to a physical machine

How do VMs work?

hypervisor = layer between VM and kernel

emulates system calls

allows multiple types of operating systems on a machine
(Windows on Linux)

overhead for hypervisor

Containers on the other hand ...

only contain application and application-related libraries and frameworks, that run on the host machine's kernel

smaller

lower overhead

differences in OS distributions and dependencies are abstracted - same kernel



Working together, not against each other

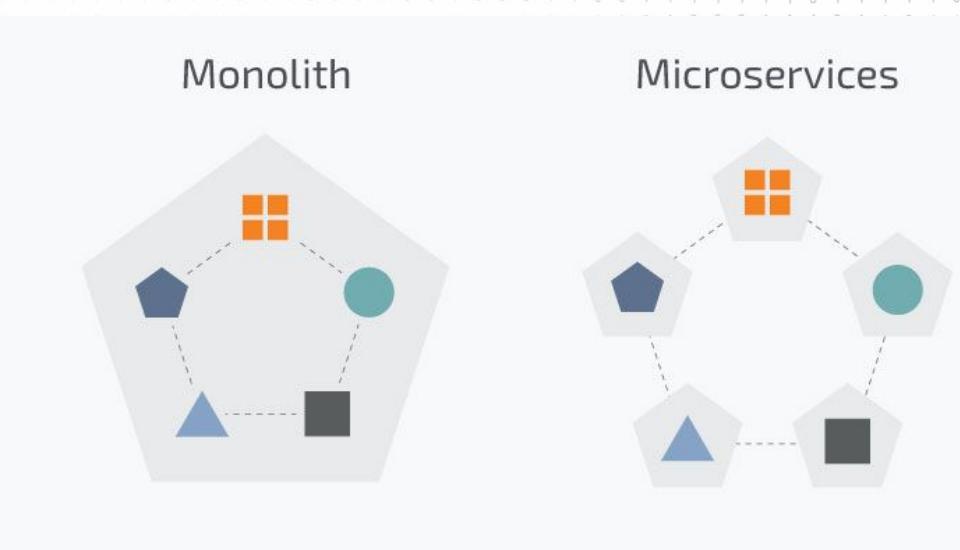
Windows on Linux possible only with VMs

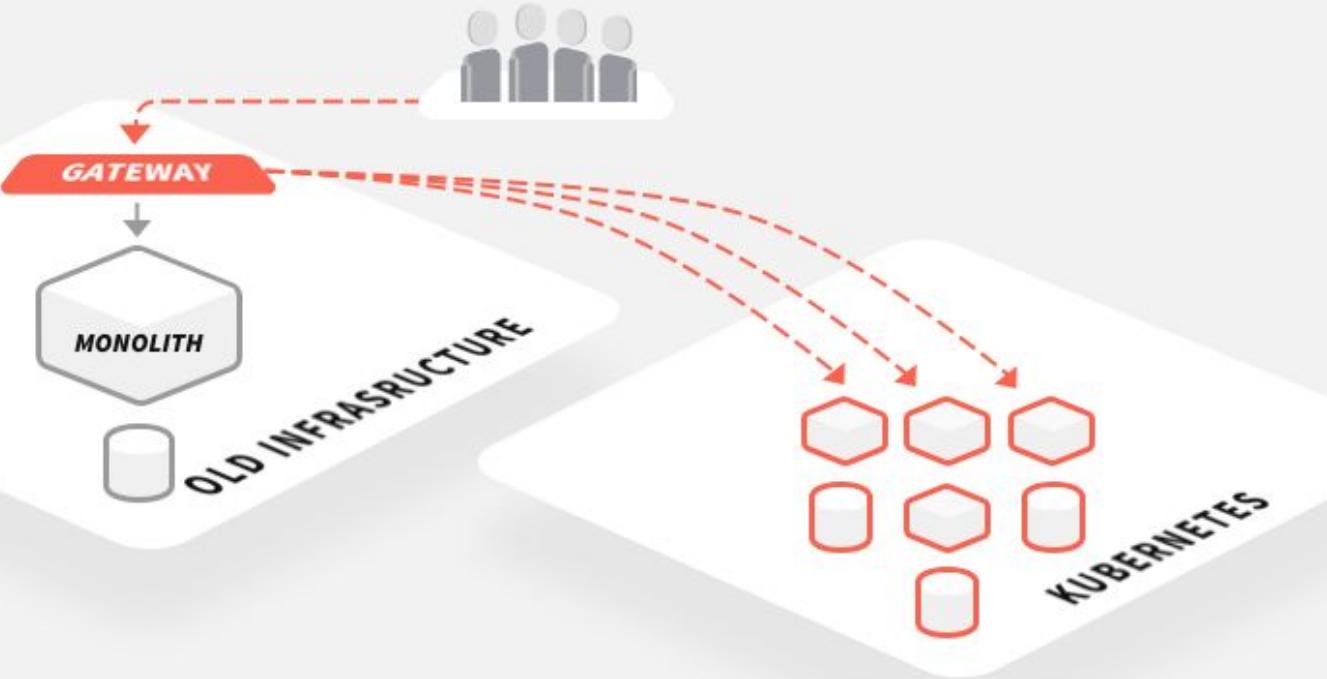
older software needs to be adapted to be run as containers (and won't)

usage of VMs as a medium for containers (better isolation and easier scaling)

Greater modularity in software

Monolithic application → independent services that interact
(microservices)





Containers empowering microservices

quicker start times -> easy to prototype or scale

allow work to be done independently on modules -> independent releases for components (take care of interfaces)

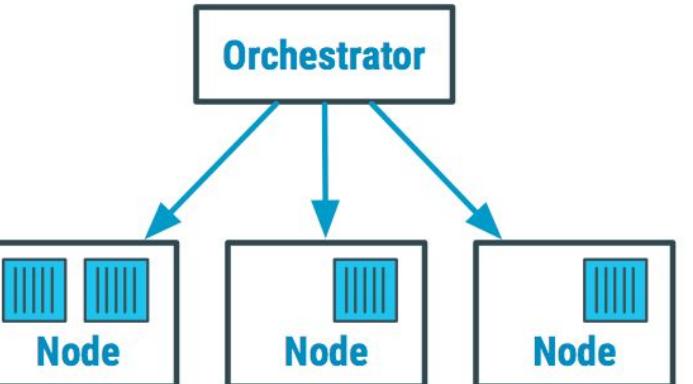
isolated and abstracted runtime environments, that can be tailored for each module

shared runtime environment, for heterogeneous applications

Orchestrator

manage and organize both hosts and docker containers running on a cluster

main issue - resource allocation - where can a container be scheduled, to fulfill its requirements (CPU/RAM/disk) + how to keep track of nodes and scale



What is Kubernetes?



kubernetes

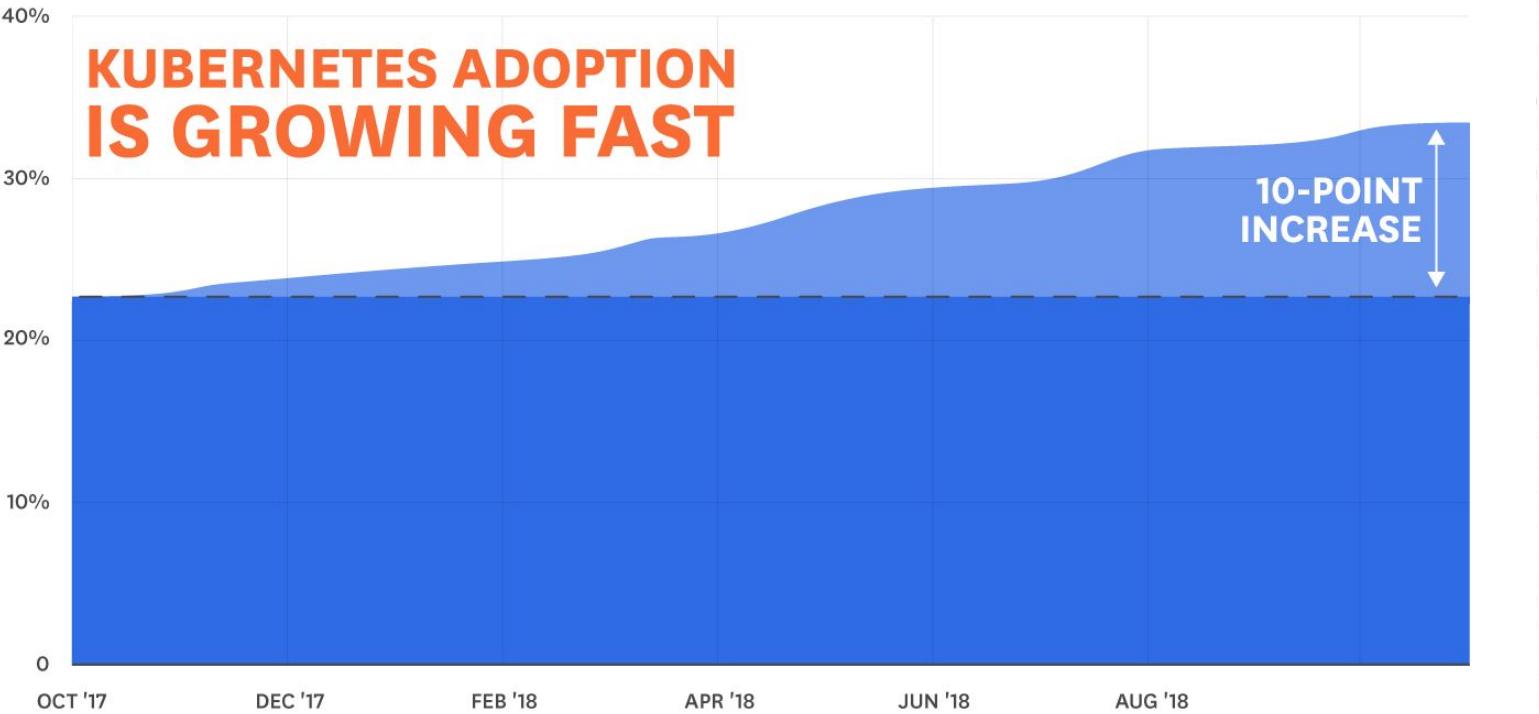
“Kubernetes” = Greek for governor, helmsman, captain

open-source container orchestration system

originally designed by Google, maintained by CNCF

aim to provide "platform for automating deployment, scaling and operations of application containers across clusters of hosts"





Pod

smallest deployable unit of computing in Kubernetes

colocated multiple apps(containers) into a single atomic unit,
scheduled onto a single machine

upon creation, statically allocated to a certain node

Communication challenges

between pods - using hardcoded IPs would be the wrong way to do it, as pods might be rescheduled on different nodes and change IPs

from outside - keep track of all pods that provide a certain service and load balancer between them

Service discovery

find which processes are listening at which addresses for which services

do it quickly and reliably, with low-latency, storing richer definitions of what those services are

public DNS isn't dynamic enough to deal with the amount of updates

Service

abstraction which defines a logical set of Pods (selected using label selector), that provide the same functionality (same microservice)

different types, for different types of exposure provided by the service

Soooo many things to configure :(

at least one controller

some services

some configMaps and Secrets

preallocate persistentVolumes or create storage class for
dynamic provisioning

Solution: another level of abstraction

higher-level controller that can manage lower-level elements

for the moment, not included in Kubernetes ... YET!

BUT can be added, through third-party controllers

What is Helm?



package manager for Kubernetes

provides higher-level abstraction (Chart) to configure full-fledged applications

manage complexity, easy upgrades, simple sharing of full application setups, safe rollbacks



How does Helm work?

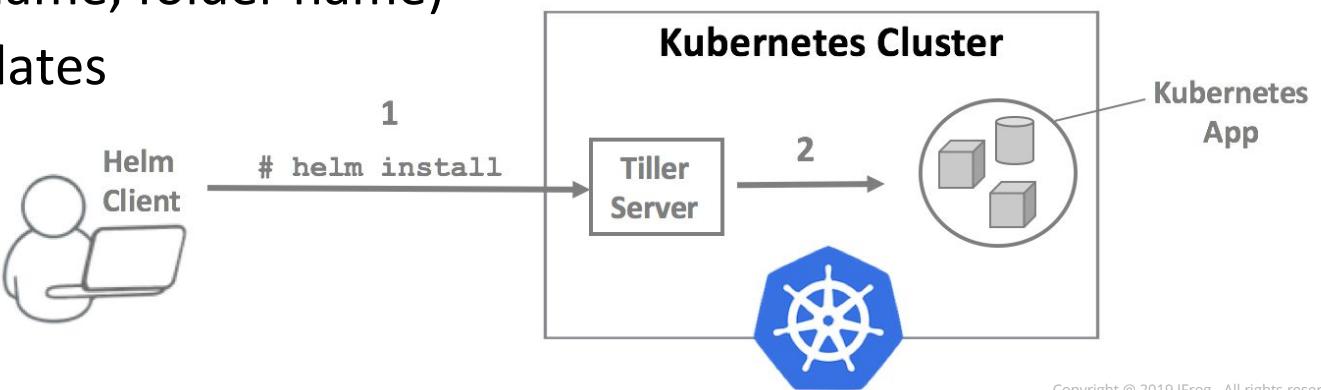
Helm CLI + Tiller server in Kubernetes (which is a controller)

CLI responsible for management + requests for releases of charts
on Kubernetes

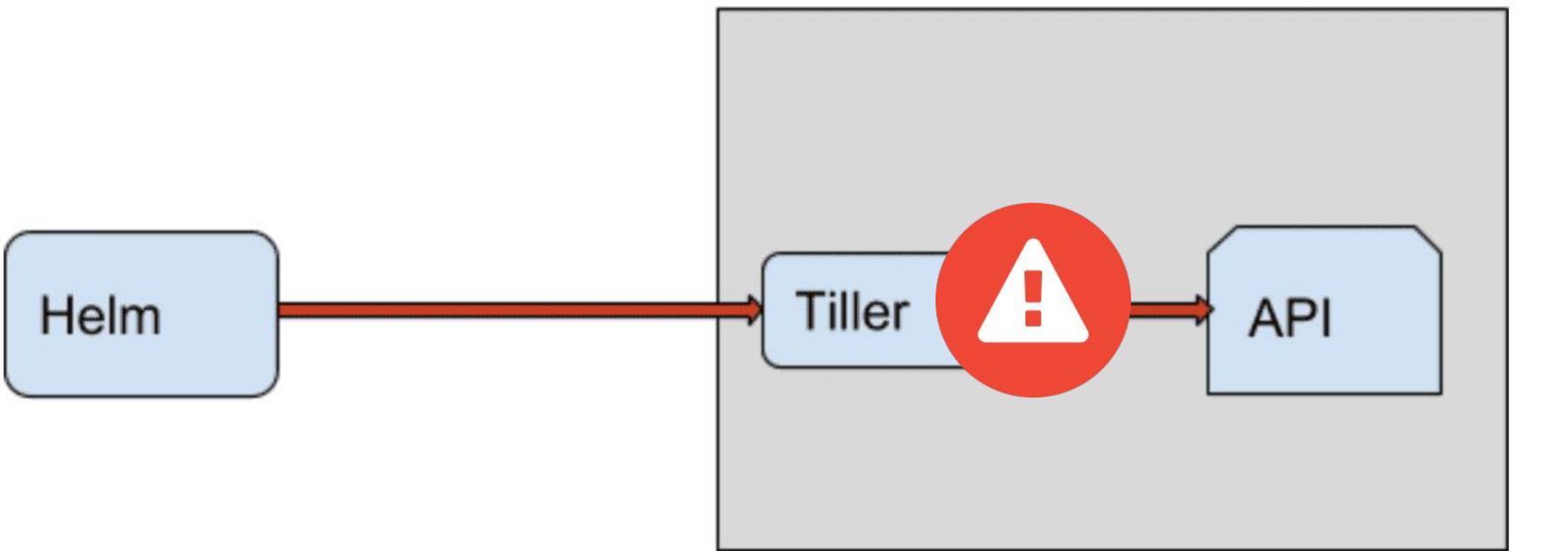
Tiller - listens for requests, combines chart + configuration =
release, install release, track release

What are the benefits of using Helm?

- The Package Manager for Kubernetes
- CLI (helm) and Server component (tiller)
- Wrapper script (Context, Application lifecycle management)
- Dependency management
- Version management (perform upgrades/rollbacks)
- Conventions (filename, folder name)
- Add logic to templates



Kubernetes cluster

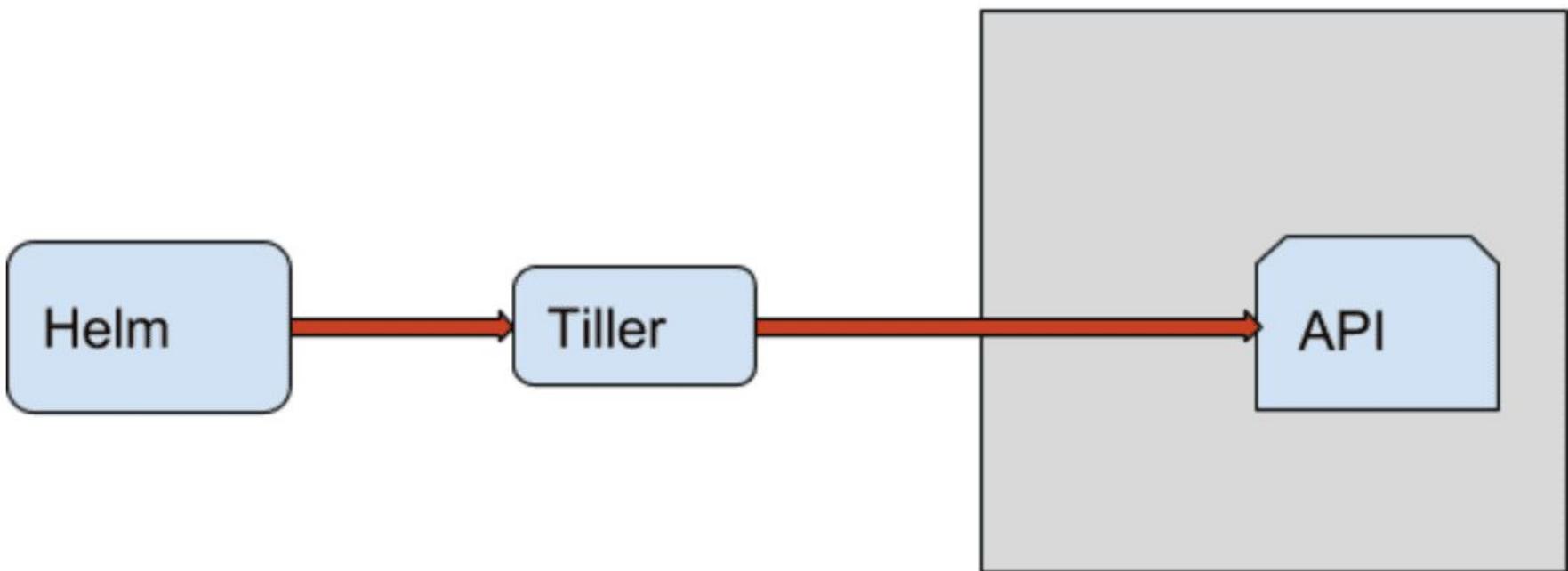




Tiller security issues

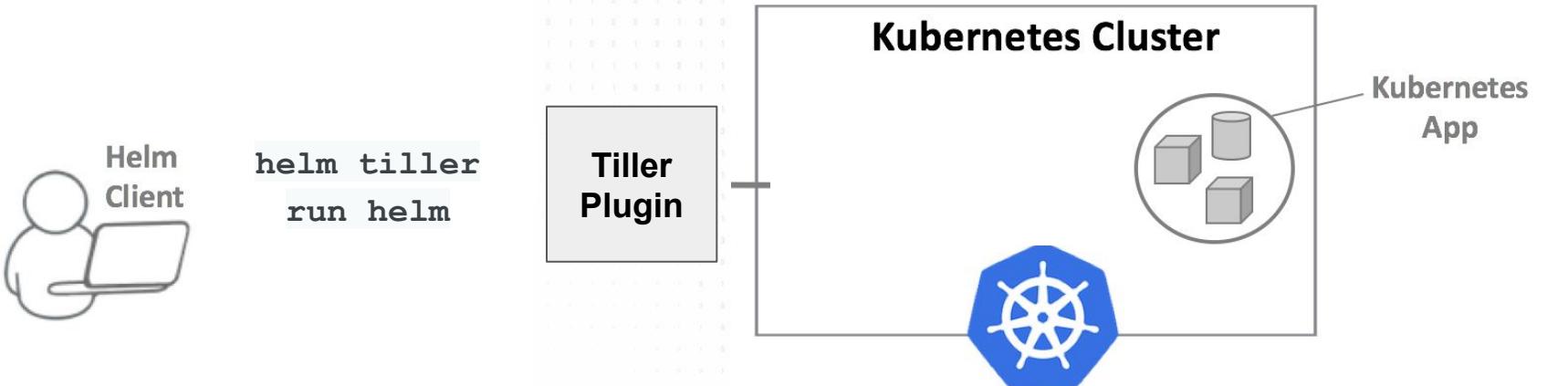
The main security issue is that tiller runs in your Kubernetes cluster with full administrative rights which is a risk if somebody gets unauthorised access to the cluster

Kubernetes cluster



from the SSH Server , run :

helm plugin install <https://github.com/rimusz/helm-tiller>



helm tiller run -- helm ls

Installed Helm version v2.13.0

Installed Tiller version v2.13.0

Helm and Tiller are the same version!

Starting Tiller...

Tiller namespace: kube-system

Running: **helm ls**

----- *COMMAND OUTPUT* -----

----- *COMMAND OUTPUT* -----

Stopping Tiller...



SETUP ENV



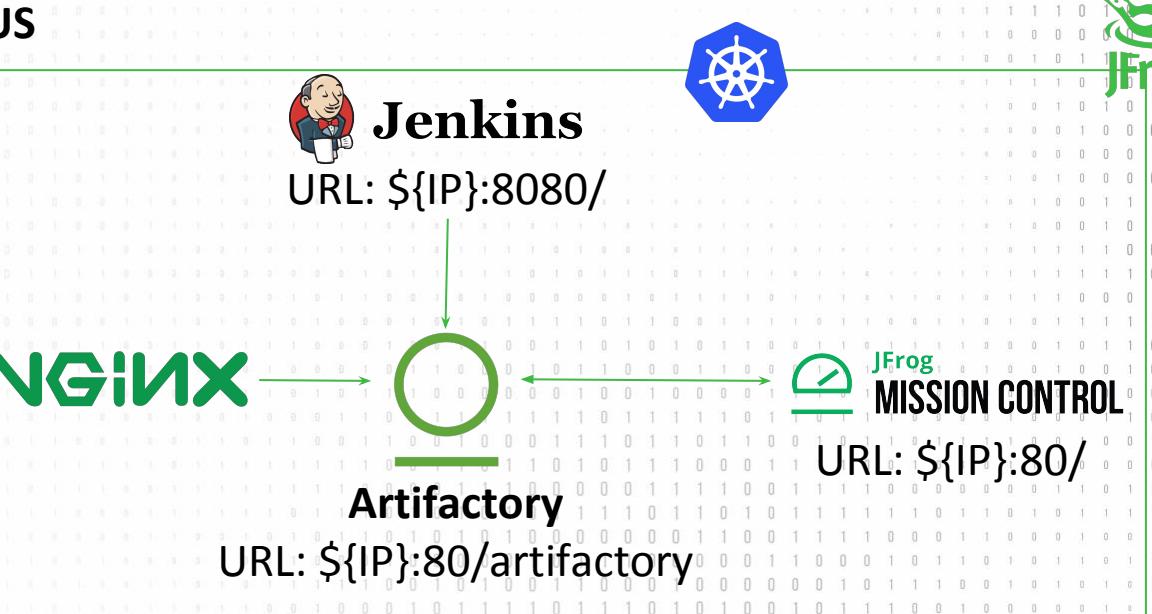
OUR DEMO SETUP



Client



Dev Server
SSH: Port 2222



IN



Setup Env instructions

```
ssh -p 2222 root@x.x.x.x
```

password - root



SETUP INSTRUCTIONS

- Run `kubectl get svc:`

- Locate **artifactory-artifactory-nginx** service **EXTERNAL-IP** (**User** : admin ,**Password** : password)
- Locate **jenkins-my-bloody-jenkins** service **EXTERNAL-IP**, (**User** : admin ,**Password** : password) , port: **8080**
- Locate **artifactory-apm-glowroot** service **EXTERNAL-IP** (no user/password needed)
-





CLASS ENVIRONMENT - SERVICES

Run `kubectl get svc`

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
artifactory-apm-cassandra	ClusterIP	10.7.240.223	<none>	9042/TCP,9160/TCP	8m
artifactory-apm-cassandra-headless	ClusterIP	None	<none>	7000/TCP,7001/TCP,7199/TCP,9042/TCP,9160/TCP	8m
artifactory-apm-glowroot	LoadBalancer	10.7.246.62	35.238.159.220	80:30414/TCP,8181:30928/TCP	8m
artifactory-artifactory	ClusterIP	10.7.241.198	<none>	8081/TCP	16m
artifactory-artifactory-nginx	LoadBalancer	10.7.240.3	35.225.251.225	80:30160/TCP,443:31396/TCP	16m
artifactory-postgresql	ClusterIP	10.7.253.239	<none>	5432/TCP	16m
jenkins-my-bloody-jenkins	LoadBalancer	10.7.243.58	104.197.138.243	8080:31013/TCP,50000:31611/TCP,16022:31813/TCP	10m
kubernetes	ClusterIP	10.7.240.1	<none>	443/TCP	18m
mission-control	LoadBalancer	10.7.248.201	104.154.178.170	80:31458/TCP,9300:30907/TCP	16m
mission-control-postgresql	ClusterIP	10.7.253.226	<none>	5432/TCP	16m
platform-sshd-dev	LoadBalancer	10.7.255.188	35.202.186.82	2222:31267/TCP	16m
sonarqube-server-postgresql	ClusterIP	10.7.243.200	<none>	5432/TCP	16m
sonarqube-server-sonarqube	LoadBalancer	10.7.251.123	35.232.120.185	9000:30637/TCP	16m



CLASS ENVIRONMENT - PODS

Run `kubectl get pods`

NAME	READY	STATUS	RESTARTS	AGE
artifactory-apm-cassandra-0	1/1	Running	0	12m
artifactory-apm-glowroot-78cccd94c4-7bq7d	1/1	Running	3	12m
artifactory-artifactory-0	1/1	Running	0	8m
artifactory-artifactory-nginx-7cc965b947-4qj21	1/1	Running	0	9m
artifactory-postgresql-7497686b49-2hc7h	1/1	Running	0	21m
jenkins-my-bloody-jenkins-67f95fb765-2pjx9	1/1	Running	0	15m
mission-control-0	5/5	Running	0	21m
mission-control-postgresql-6d6b47f775-b5fgw	1/1	Running	0	21m
platform-sshd-dev-7bd8675fb6-hmqq9	1/1	Running	0	21m
sonarqube-server-postgresql-5d76ddcb75-2k558	1/1	Running	0	21m
sonarqube-server-sonarqube-5ccc686887-6jc7q	1/1	Running	2	21m

COMMON COMMANDS

- View all K8S services/pods

```
kubectl get svc / kubectl get pods
```

- Get logs for artifactory pod

```
kubectl logs -f pod_name --container container_name
```

- SSH to Artifactory pod

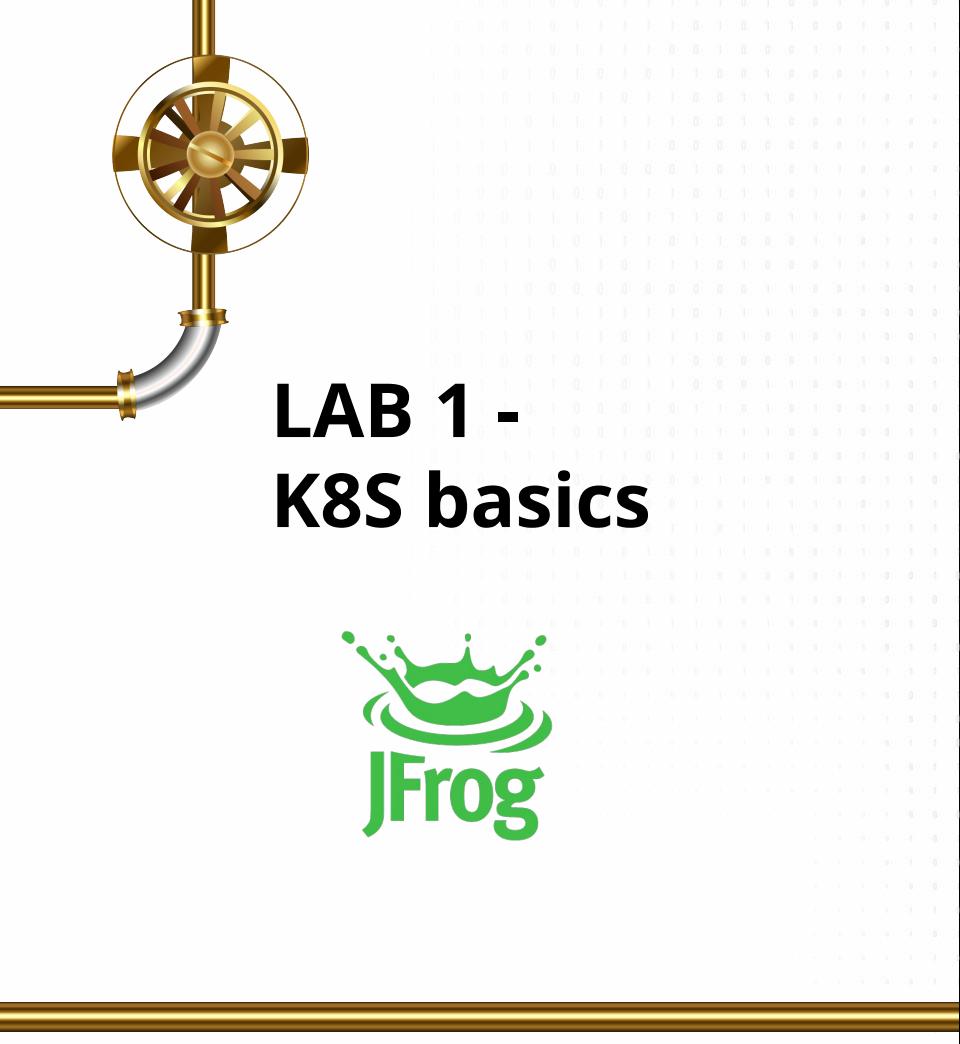
```
kubectl exec -ti pod_name -- /bin/bash
```

- Change Artifactory pod

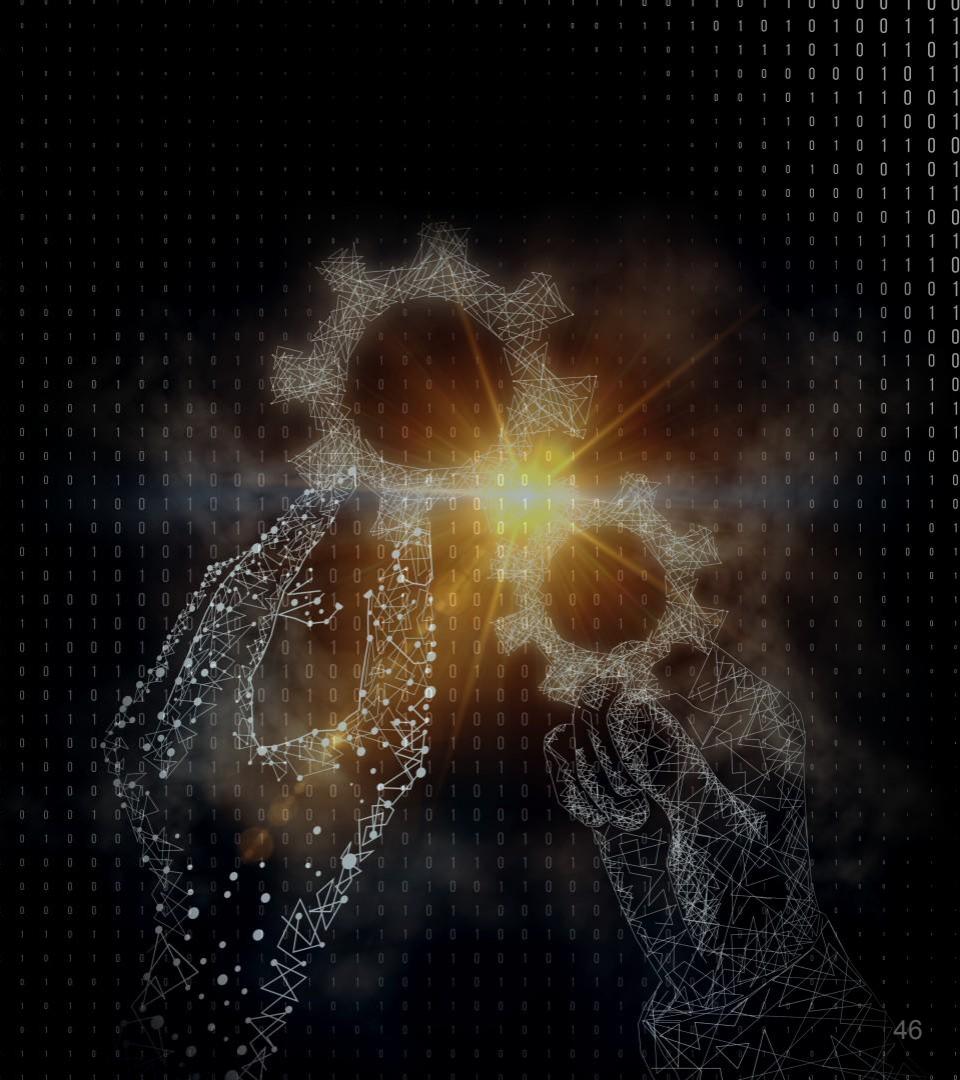
```
helm upgrade release_name jfrog/artifactory --set  
artifactory.xxxxxxx=value
```

- Install new Artifactory instance

```
helm install --name release_name jfrog/artifactory
```



LAB 1 - K8S basics



LAB 1 - K8S Basics

Since the lab environment is build on top of k8s, we have prepared a small hands-on lab for getting a bit familiar with k8s:

1. Get all available pods. Which applications do we have set up?
2. Look for the pod that has the label “app=artifactory” and exec into it.
3. When inside the container, print the environment variables (Hint: env). Do you see anything interesting? Exit from the container.
4. Get all available services.
5. Try to access the ports listed in the PORT(S) column with the IPs. Which ports can you reach? using which IPs? Why?
6. Which services are available for external access? How can we know that?
7. Using kubectl, take a look at the Artifactory logs, can you see anything interesting?
8. Using kubectl, take a look at the postgresql logs.

COMMON COMMANDS

- View all K8S services/pods

```
kubectl get svc / kubectl get pods
```

- Get logs for artifactory pod

```
kubectl logs -f pod_name --container container_name
```

- SSH to Artifactory pod

```
kubectl exec -ti pod_name -- /bin/bash
```

- Change Artifactory pod

```
helm upgrade chart_name jfrog/artifactory --set  
artifactory.xxxxxxx=value
```

- Install new Artifactory instance

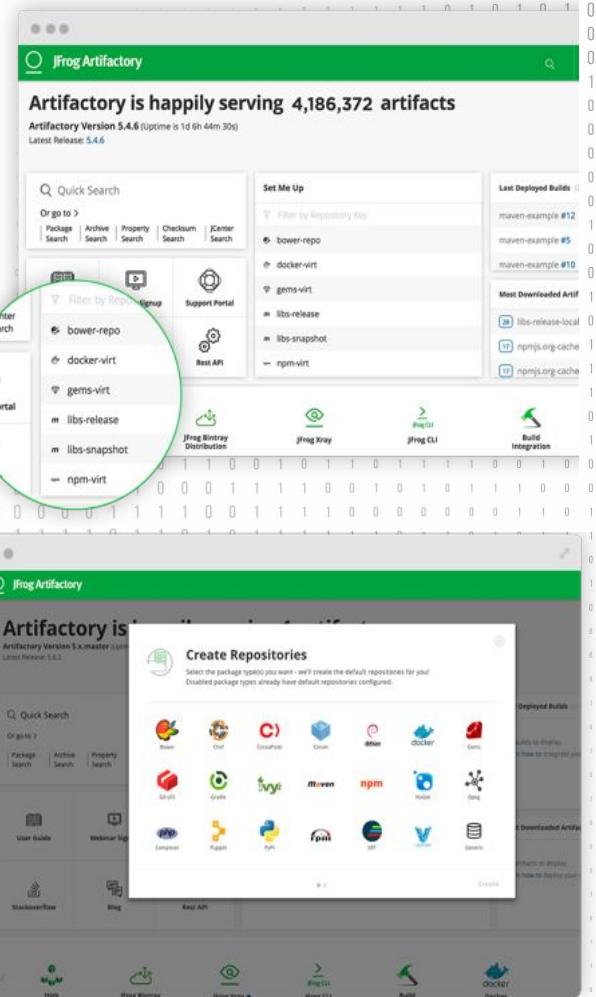
```
helm install --name chart_name jfrog/artifactory
```

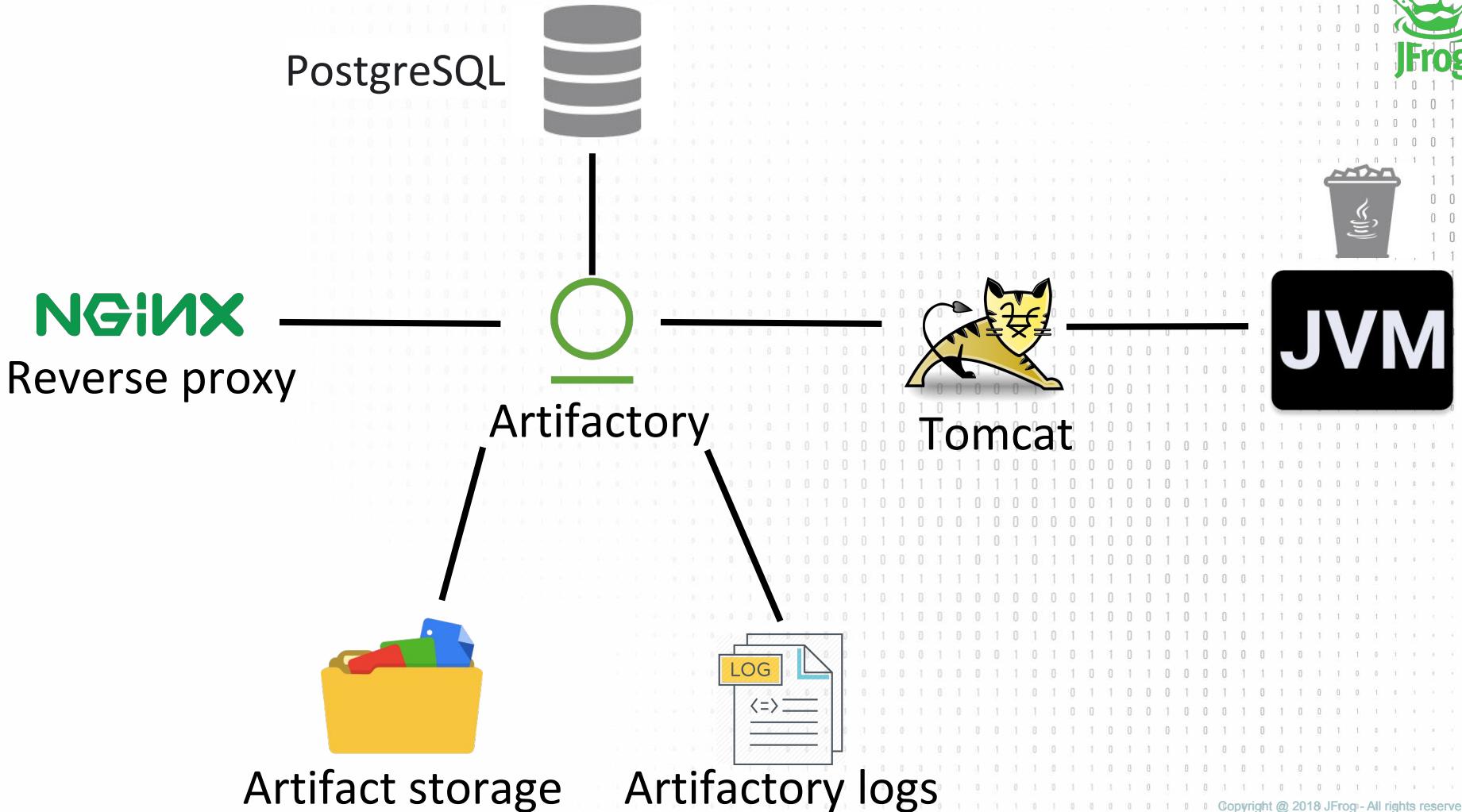
ARTIFACTORY INTERNALS



ARTIFACTORY

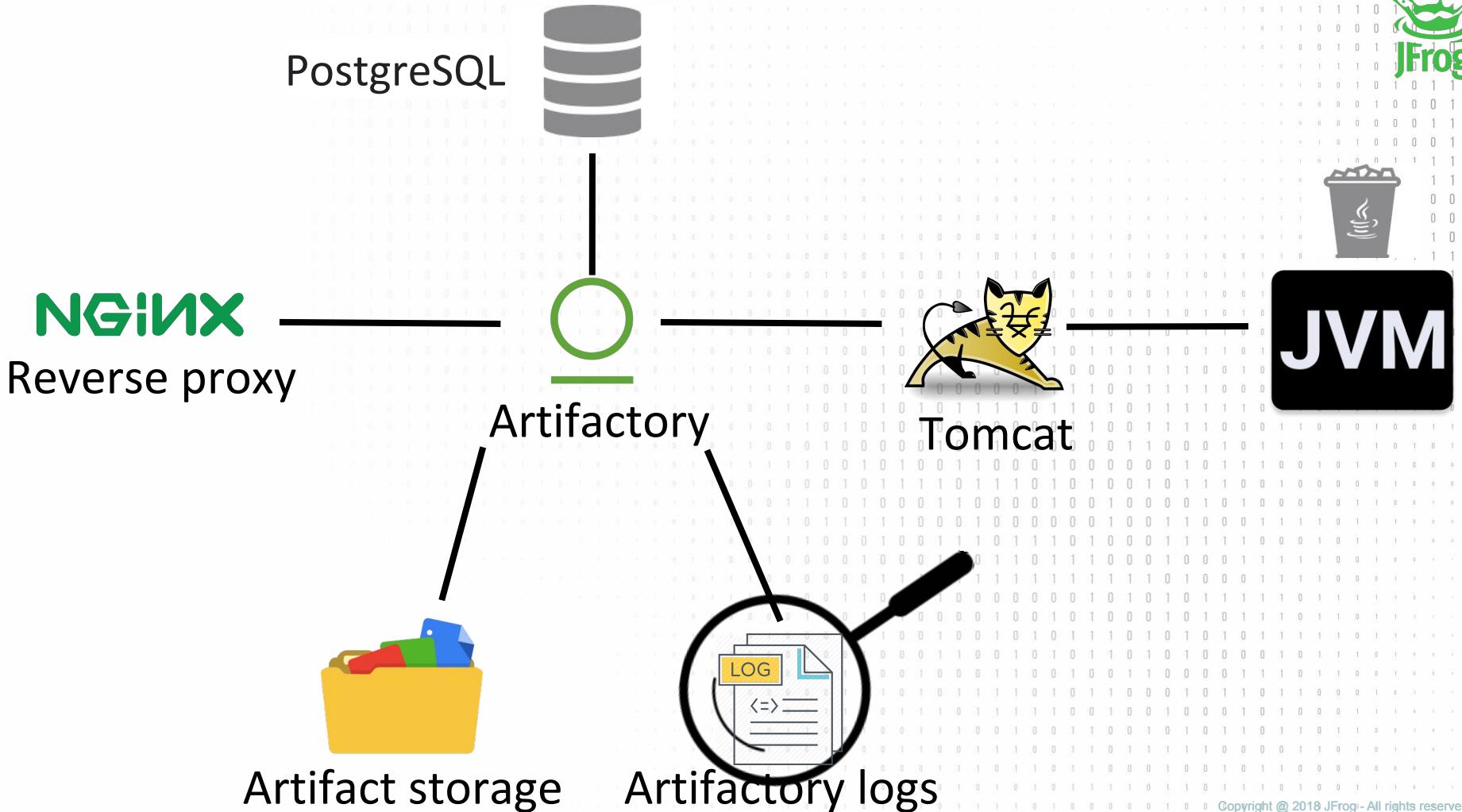
- Stores and Manages your binaries
- Universal
- Scalable
- Fully Automated
- Integrated with all major build tools and CI servers



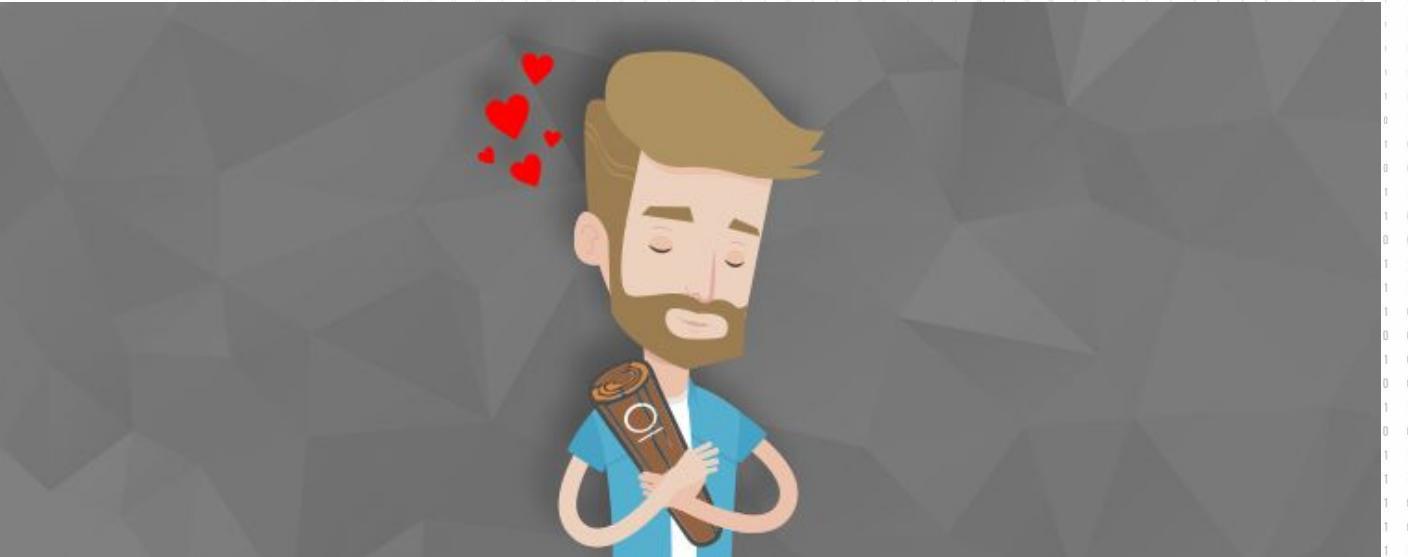


Labs details - <http://bit.ly/2XxQuHL>



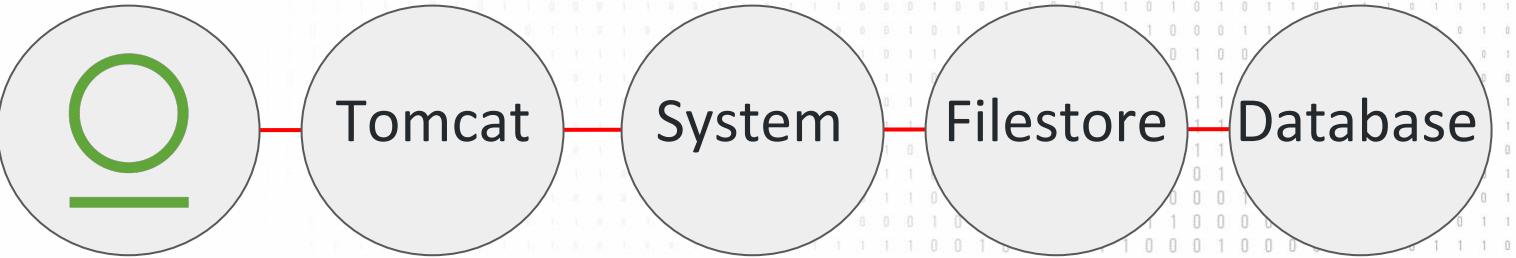


LOVE YOUR LOGS, IT PAYS OFF!



EVENT SOURCES

- Logs primary source



- Health & Readiness check available via
REST API: /api/system/ping

ARTIFACTORY LOGS: logback.xml

<code>artifactory.log</code>	The main Artifactory log file containing data on Artifactory server activity.
<code>access.log</code>	Access log containing important information about accepted and denied requests, configuration changes and password reset requests. The originating IP address for each event is also recorded.
<code>request.log</code>	Generic http traffic information similar to the Apache HTTPd request log.
<code>request_trace.log</code>	Extended information, including HTTP headers, when requests are made including <code>?trace</code>
<code>import.export.log</code>	Log used for tracking the process of long-running import and export commands.
<code>traffic.log</code>	Traffic log providing information similar to the request log, specifically for replication. Disabled by default. To enable, in the set <code>artifactory.traffic.collectionActive=true</code> in <code>artifactory.system.properties</code>





LAB1: UPDATING LOGGING LEVELS



ACCESS

- Bundled with Artifactory and runs as a separate service under the same Tomcat
- Provides a common authentication and authorization infrastructure for all of JFrog products to manage users, groups, permissions and tokens
- Different access services can be federated into a single “circle of trust”
 - Enables synchronisation of users, groups and permissions between sites rather than having to define them separately
 - Enables single-sign-on for all JFrog services



LAB1: UPDATING LOGGING LEVELS

- Send an authenticated request to Artifactory with **bad credentials**, i.e:

```
curl -u admin:wrongPassword
```

```
http://x.x.x.x/artifactory/api/system/ping
```

- Notice the 401, but nothing in the
/var/opt/jfrog/artifactory/logs/artifactory.log
- Let's use the Artifactory HttpClient debug logging
- Find the relevant communication attempt between
Artifactory<->Access, and Access return **401 response** to Artifactory

Apache HttpClient - *org.apache.http.wire*

- The wire log is used to log all data transmitted to and from servers when executing HTTP requests.
- Enable HttpClient debug logging:

```
<logger name="org.apache.http.wire">  
  <level value="debug"/>  
</logger>
```

This log should only be enabled to debug problems, as it will produce an extremely large amount of data



LAB2 SETUP: HELP, I'VE GOT 1GB DISK LEFT!





PostgreSQL



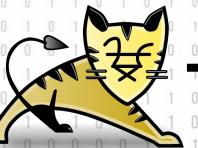
NGINX

Reverse proxy

Artifactory



Artifact storage



Tomcat



Artifactory logs

LAB2: HELP, I'VE GOT 1GB DISK LEFT!

OK, not really 1GB disk left, but, let's create a lot of extra artifacts

- Go to Jenkins <http://xxx.xxx.xxx.xxx:8080>
- Click on the **generate-packages-job**
- Run once to trigger Job Param
- The build will fail
- Click once on “Build with Parameters”
- Create a generic repository on Artifactory
- **REPO_NAME** - The name of the repo
- Artifactory URL need to be :
http://xxx.xxx.xxx.xxx/artifactory
- Enter the parameters and click “Build”
- **Leave this job running**

Pipeline generate-packages-job

This build requires parameters:

ARTIFACTORY_URL

http://146.148.58.205/artifactory

without trailing '/'

please select artifactory url - http://xxx.xxx.xxx.xxx/artifactory

REPO_NAME

generic-local

Please select target repo name

PACKAGE_SIZE_MIN

35000000

Please select min size (bytes)

PACKAGE_SIZE_MAX

50000000

Please select max size (bytes)

NUM_OF_ARTIFACTS

35

Please select num of artifacts to generate

Build



LAB3: HIGH HTTP(S) REQUESTS



LAB3: HIGH HTTP(S) REQUESTS



This site can't be reached

34.66.185.31 refused to connect.

Try:

- Checking the connection
- [Checking the proxy and the firewall](#)

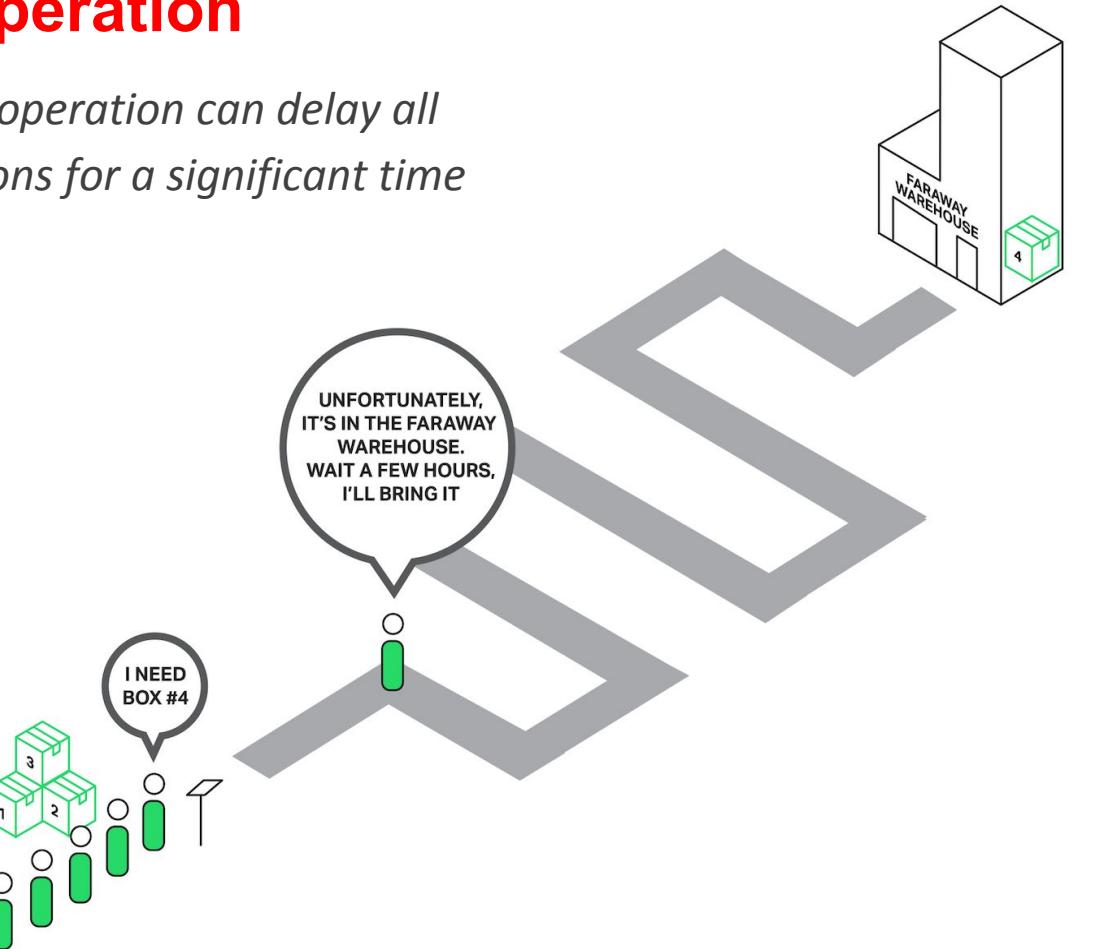
ERR_CONNECTION_REFUSED

Details

Reload

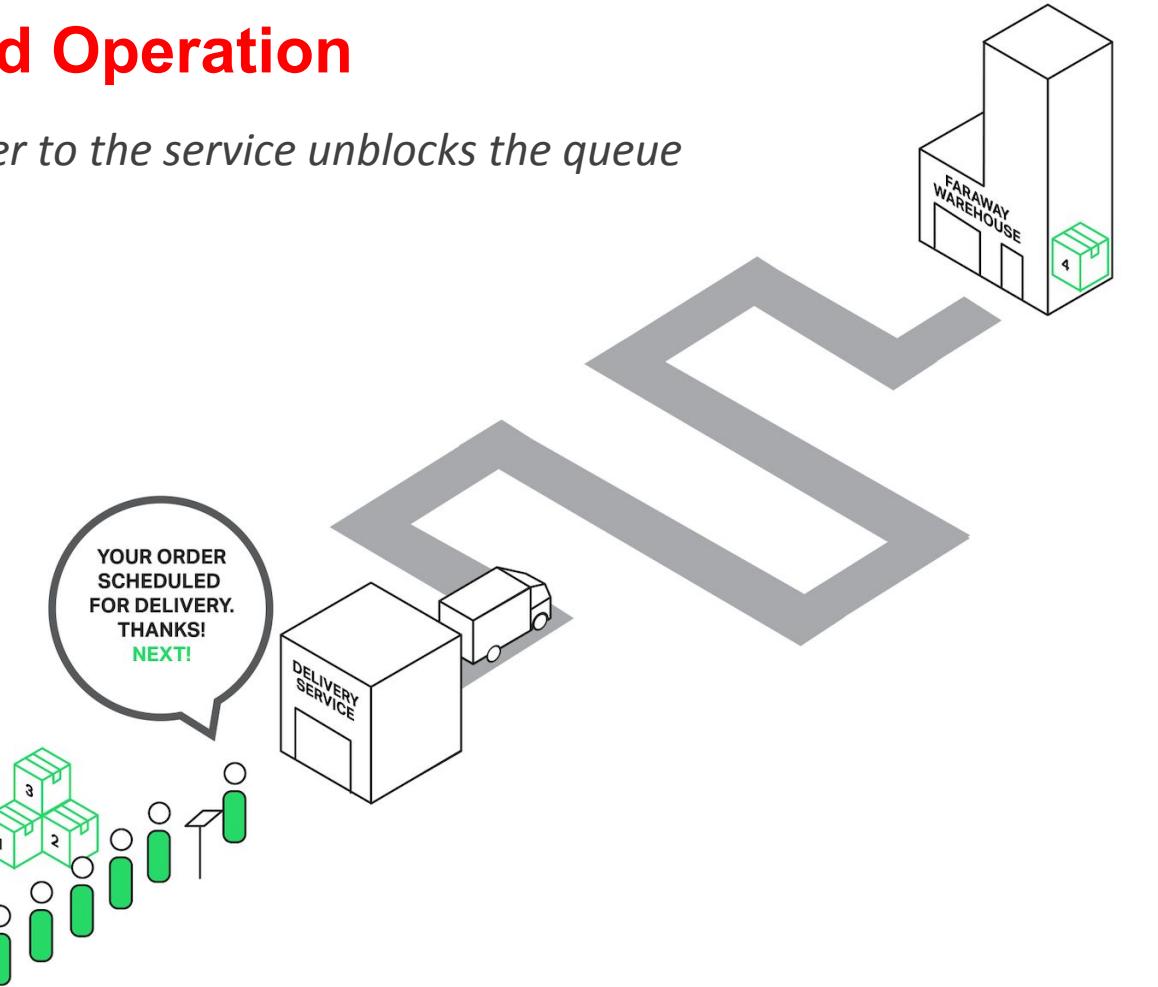
Blocking Operation

Just one blocking operation can delay all following operations for a significant time



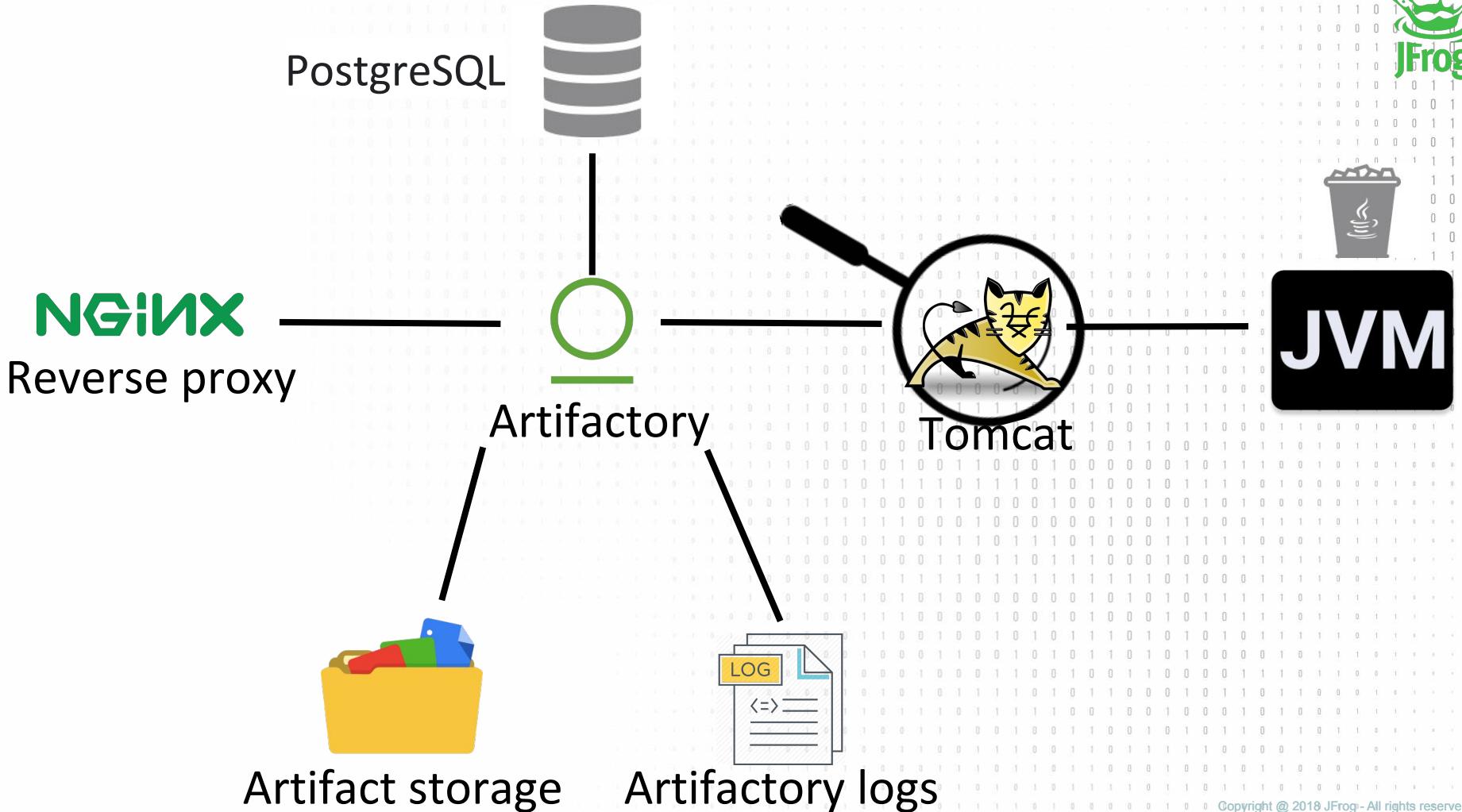
Scheduled Operation

Passing an order to the service unblocks the queue

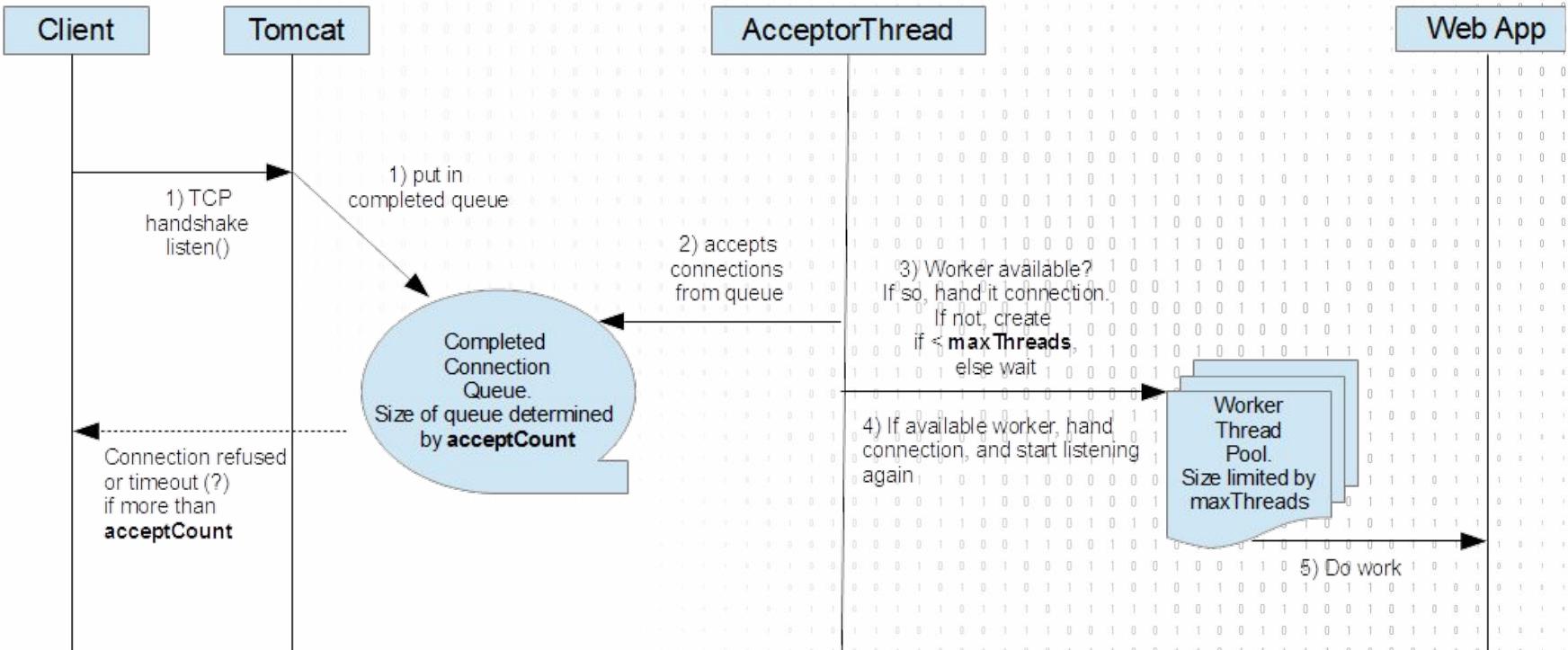


BUT STILL - HOW LONG IS THE LINE?





```
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
redirectPort="8443" acceptCount="1000" maxConnections="500" />
```



SET ARTIFACTORY MAX THREADS

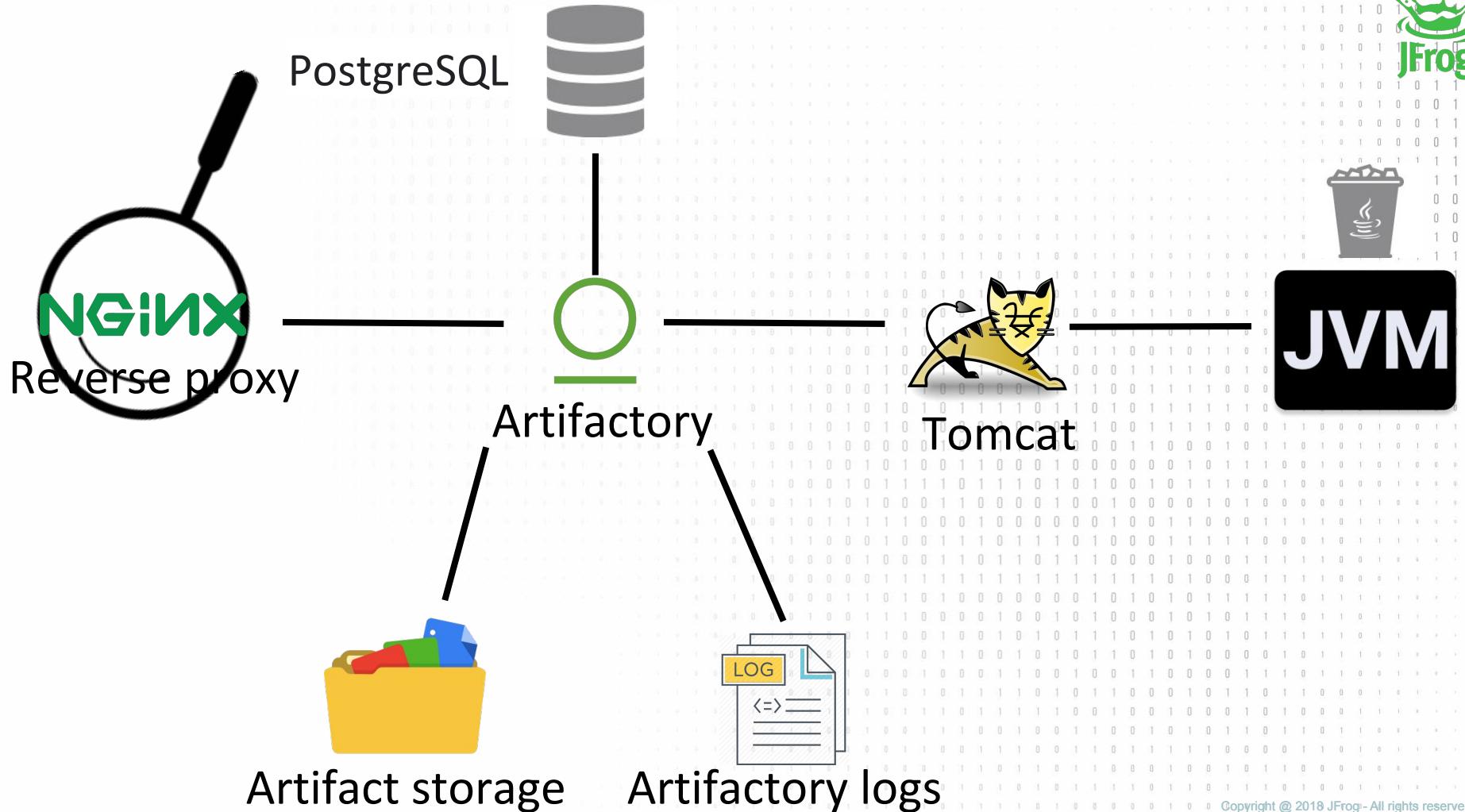
- <https://github.com/jfrog/charts/tree/master/stable/artifactory>
- <https://www.jfrog.com/confluence/display/RTF/Installing+with+Docker>
(look for SERVER_XML_ARTIFACTORY_MAX_THREADS)

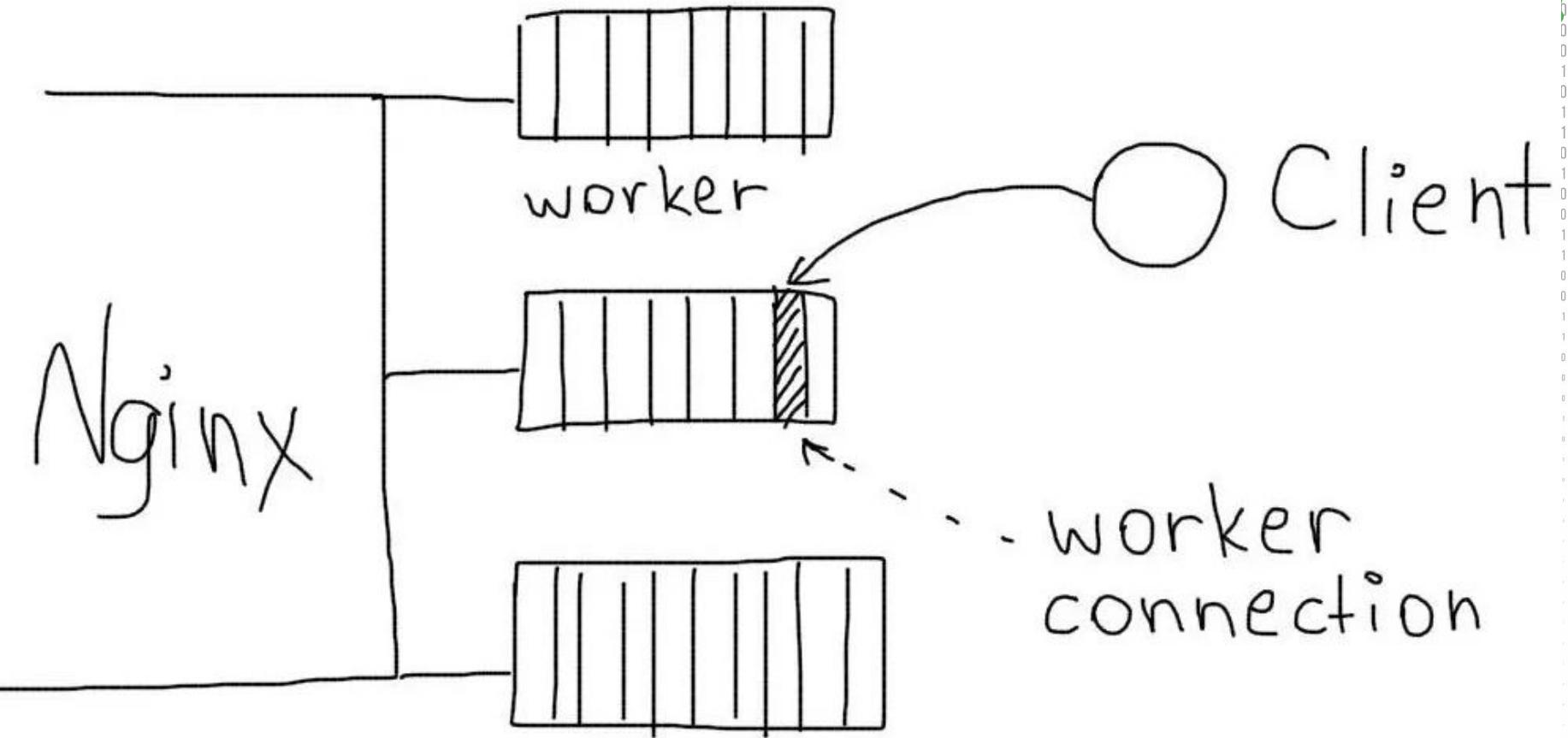
```
helm upgrade artifactory jfrog/artifactory  
--version 7.17.1 -f artifactory.yaml
```

artifactory:

extraEnvironmentVariables:

- name: SERVER_XML_ARTIFACTORY_EXTRA_CONFIG
value: 'maxConnections="2" acceptCount="1"'
- name: SERVER_XML_ARTIFACTORY_MAX_THREADS
value: "2"





SET NGINX WORKER CONNECTION

- Create a configMap: `kubectl create configmap nginx-conf --from-file=nginx.conf`
- reload artifactory: `helm upgrade artifactory jfrog/artifactory --version 7.17.1 --set nginx.customConfigMap=nginx-conf`
- How it's affecting our server - try running ab command to simulate traffic
- Change **worker_connections** by increasing this value, the hope is that we can increase the capacity of each worker process

LAB4: DATABASE CONNECTIONS EXCEEDED





PostgreSQL



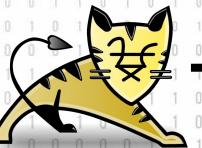
NGINX
Reverse proxy

Artifactory



Artifact storage

Artifactory logs



Tomcat



LAB4: DATABASE CONNECTIONS EXCEEDED



- Change Postgres settings:

```
helm upgrade artifactory jfrog/artifactory
```

```
    --version 7.17.1
```

```
    --set postgresql.postgresPassword=zooloo
```

```
    --set postgresql.postgresConfig.max_connections=1
```

```
    --set
```

```
    postgresql.postgresConfig.superuser_reserved_connections=0
```

- Go to our Artifactory instance

- Notice that it hangs and doesn't load completely

LAB4: DATABASE CONNECTIONS EXCEEDED



- In the **artifactory.log** file you should see something like this:
 - Caused by: `org.postgresql.util.PSQLException`:
Data source rejected establishment of connection, message
from server: "Too many connections"
- Can we resolve this issue by adjusting Artifactory configuration only?
<https://jfrog.com/knowledge-base/how-do-i-tune-artifactory-for-heavy-loads/>





LAB2 PHASE II: HELP, I'VE GOT 1GB DISK LEFT!





PostgreSQL



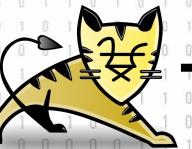
NGINX

Reverse proxy

Artifactory



Artifact storage



Tomcat



Artifactory logs

LAB2: HELP, I'VE GOT 1GB DISK LEFT!

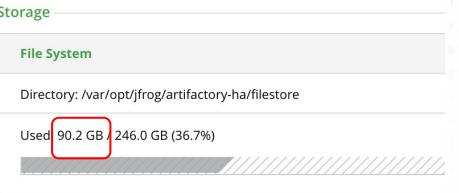
- If you have implemented a promotion policy you might be able to delete your snapshot repo or equivalent in entirety
- Take a more proactive approach to managing it using a combination of plugins, AQL and the CLI
- Schedule deletes



LAB2: HELP, I'VE GOT 1GB DISK LEFT!

- Browse to the **repo we created** earlier, right click on it and click “Delete Content”
- Keep the storage window open before you delete, and open a new one after
- As you can see - the **storage usage is the same** ever after we delete content **before**
after

Binaries Size:	29.29 GB	Binaries Count:	1,500
Artifacts Size:	39.61 GB	Artifacts Count:	2,567
Optimization:	73.95%	Items Count:	3,723

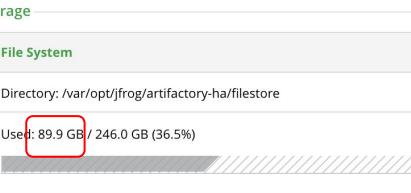


66 Repositories [?](#)

Filter by Repository Key

Reposit...	Reposit...	Packag...	Percent...	Artifac...	Files
TOTAL	N/A	N/A	N/A	39.61 GB	2567
Trash C...	N/A	Trash	6.71%	2.66 GB	146
docker-s...	LOCAL	Doc...	48.87%	19.35 GB	1029

Binaries Size:	29.30 GB	Binaries Count:	1,555
Artifacts Size:	61.32 GB	Artifacts Count:	3,703
Optimization:	47.78%	Items Count:	4,907



66 Repositories [?](#)

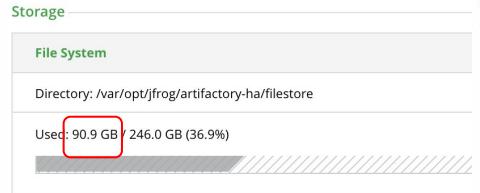
Filter by Repository Key

Reposit...	Reposit...	Packag...	Percent...	Artifac...	Files
TOTAL	N/A	N/A	N/A	61.32 GB	3703
Trash Can	N/A	Trash	39.74%	24.37 GB	1282
docker-s...	LOCAL	Doc...	31.56%	19.35 GB	1029

LAB2: HELP, I'VE GOT 1GB DISK LEFT!

- Browse to the **repo we created** earlier, right click on it and click “Delete Content”
- Keep the storage window open before you delete, and open a new one after
- As you can see - the **storage usage is the same** ever after we delete content **before** **after**

Binaries Size:	29.06 GB	Binaries Count:	1,527
Artifacts Size:	60.85 GB	Artifacts Count:	3,631
Optimization:	(?) 47.75%	Items Count:	(?) 4,836

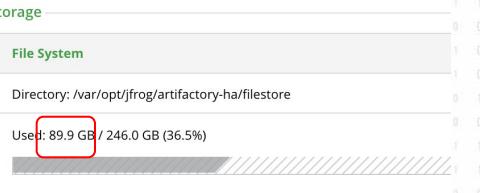


66 Repositories (7)

Filter by Repository Key

Reposit...	Reposit...	Packag...	Percent...	Artifac...	Files
TOTAL	N/A	N/A	N/A	60.85 GB	3631
Trash C...	N/A	Trash	4%	2.43 GB	984
debian...	LOCAL	Deb...	35.67%	21.71 GB	227

Binaries Size:	29.30 GB	Binaries Count:	1,555
Artifacts Size:	61.32 GB	Artifacts Count:	(?) 3,703
Optimization:	(?) 47.78%	Items Count:	(?) 4,907

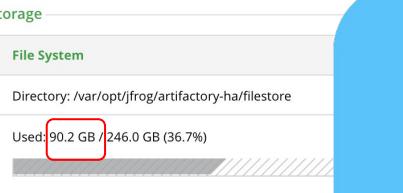


66 Repositories (7)

Filter by Repository Key

Reposit...	Reposit...	Packag...	Percent...	Artifac...	Files
TOTAL	N/A	N/A	N/A	61.32 GB	3703
Trash Can	N/A	Trash	39.74%	24.37 GB	1282
docker-s...	LOCAL	Doc...	31.56%	19.35 GB	1029

Binaries Size:	29.29 GB	Binaries Count:	1,500
Artifacts Size:	39.61 GB	Artifacts Count:	(?) 2,567
Optimization:	(?) 73.95%	Items Count:	(?) 3,723



66 Repositories (7)

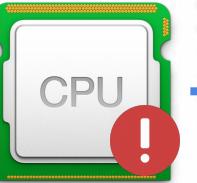
Filter by Repository Key

Reposit...	Reposit...	Packag...	Percent...	Artifac...	Files
TOTAL	N/A	N/A	N/A	39.61 GB	2567
Trash C...	N/A	Trash	6.71%	2.66 GB	146
docker-s...	LOCAL	Doc...	48.87%	19.35 GB	1029



ARTIFACTORY - GARBAGE COLLECTION

- When an Artifactory user "deletes" a file, what is actually deleted is the reference from the Artifactory database to the physical file.
- Before actually deleting a file Artifactory ensure that there are no other references to the file.



Scanning the system
is very CPU intensive



Schedule based
“Garbage Collection”

LAB2: HELP, I'VE GOT 1GB DISK LEFT!

- Now that we know what GC does, let's run it by going to:

WebUI → Admin → Advanced → Maintenance → Garbage Collection → Run

before

Binaries Size:	29.29 GB	Binaries Count:	1,500
Artifacts Size:	39.61 GB	Artifacts Count:	2,567
Optimization:	73.95%	Items Count:	3,723

Storage

File System

Directory: /var/opt/jfrog/artifactory-ha/filestore

Used: 90.2 GB / 246.0 GB (36.7%)

66 Repositories

Filter by Repository Key

Reposit...	Reposit...	Packag...	Percent...	Artifac...	Files
TOTAL	N/A	N/A	N/A	39.61 GB	2567
Trash C...	N/A	Trash	6.71%	2.66 GB	146
docker-...	LOCAL	Doc...	48.87%	19.35 GB	1029

after

Binaries Size:	8.25 GB	Binaries Count:	1,128
Artifacts Size:	40.27 GB	Artifacts Count:	2,578
Optimization:	20.48%	Items Count:	3,737

Storage

File System

Directory: /var/opt/jfrog/artifactory-ha/filestore

Used: 70.0 GB / 246.0 GB (28.5%)

66 Repositories

Filter by Repository Key

Reposit...	Reposit...	Packag...	Percent...	Artifac...	Files
TOTAL	N/A	N/A	N/A	40.27 GB	2578
Trash C...	N/A	Trash	6.6%	2.66 GB	146
docker-...	LOCAL	Doc...	49.72%	20.02 GB	1040

STORAGE RELATED REST API

- Delete artifact (REST API) :
 - DELETE `http://xxx.xxx.xxx.xxx/artifactory/repoName/path`
- Empty Trash Can (REST API) :
 - DELETE `/api/trash/clean/repoName/path`

Advanced cleanup using AQL and JFrog CLI

<https://www.jfrog.com/confluence/display/RTF/Artifactory+Query+Language>



Let's build an AQL query that identifies candidate artifacts
for cleanup with **size > 45 MB**
and invoke it with **JFrog CLI**



DATA COLLECTION (MONITORING) POINTS



FAULT - EVENT SOURCES

- REST API: /api/system/ping
- Provides a complete system health check
 - Returns an HTTP 200 code with “OK” if Artifactory is running
 - Otherwise returns a 5xx with stated reason
 - Checks to see if socket is open, and if full Artifactory service including DB and filestore is functional



MONITORING REST API

Name	Usage	Description	Produces
Scheduled Replication Status	GET <code>/api/replication/{repoKey}</code>	Returns the status of scheduled cron-based replication jobs define via the Artifactory UI on repositories. Supported by local, local-cached and remote repositories	<code>application/json</code>
Get Storage Summary Info	GET <code>/api/storageinfo</code>	Returns storage summary information regarding binaries, file store and repositories.	<code>application/json</code>
System Info	GET <code>/api/system</code>	Display different system information such as JVM runtime parameters, JVM arguments, memory usage and more.	<code>text/plain</code>



LAB5 : MONITOR ARTIFACTORY JVM



GlowRoot APM

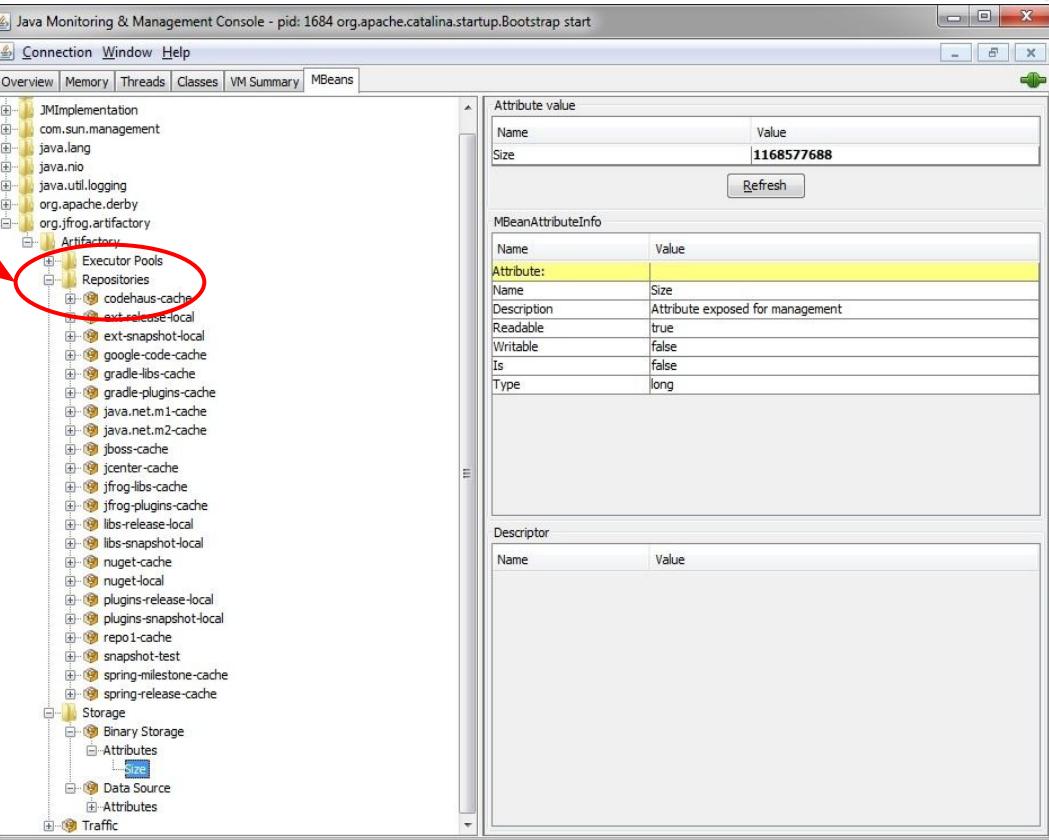
- Community driven open source monitoring for JVM based applications
- Capture errors, interact with JVM, identify bottlenecks and provide off the shelf monitoring
- Single instance or central collector configuration available
- Multi node architecture uses Cassandra for data-store with a **Docker** and **Helm** deployments available
- GlowRoot url - look for the ***artifactory-apm-glowroot*** service



ARTIFACTORY MBeans

Artifactory exposes MBeans,
under the *org.jfrog.artifactory*
domain

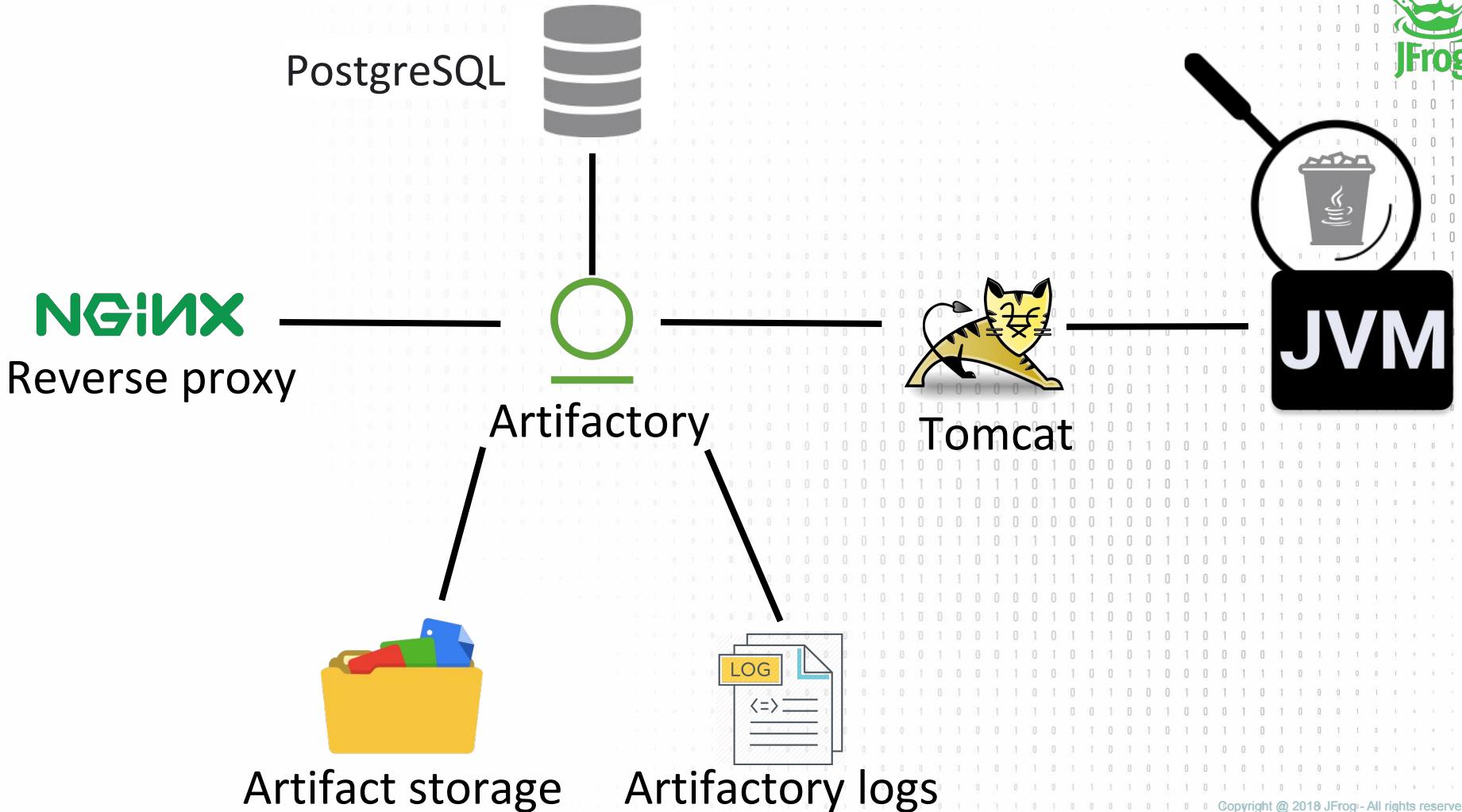
Let you monitor repositories,
executor pools, storage and
HTTP connection pools



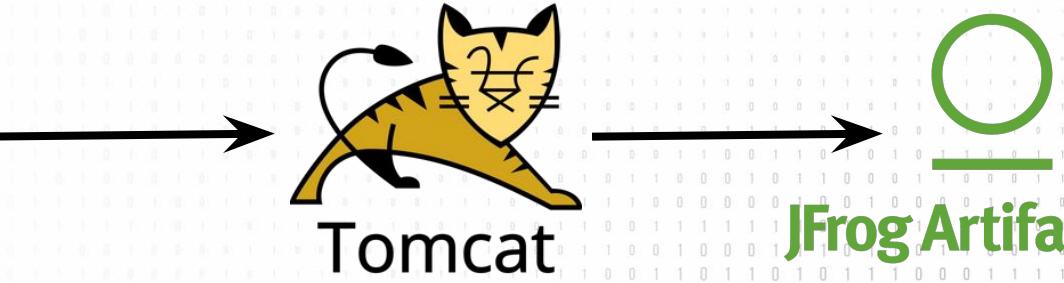
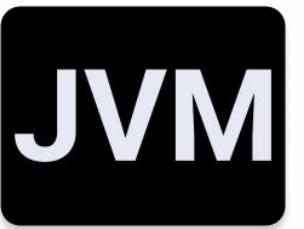


LAB5: JVM MEMORY ISSUES





TOMCAT - JVM TUNING



-server -Xms512m -Xmx2g -Xss256k -XX:+UseG1GC

By increasing your heap (Xmx) you increase bandwidth capability of Artifactory

16gb of Heap is equal to about 1GBit/s Network Speed





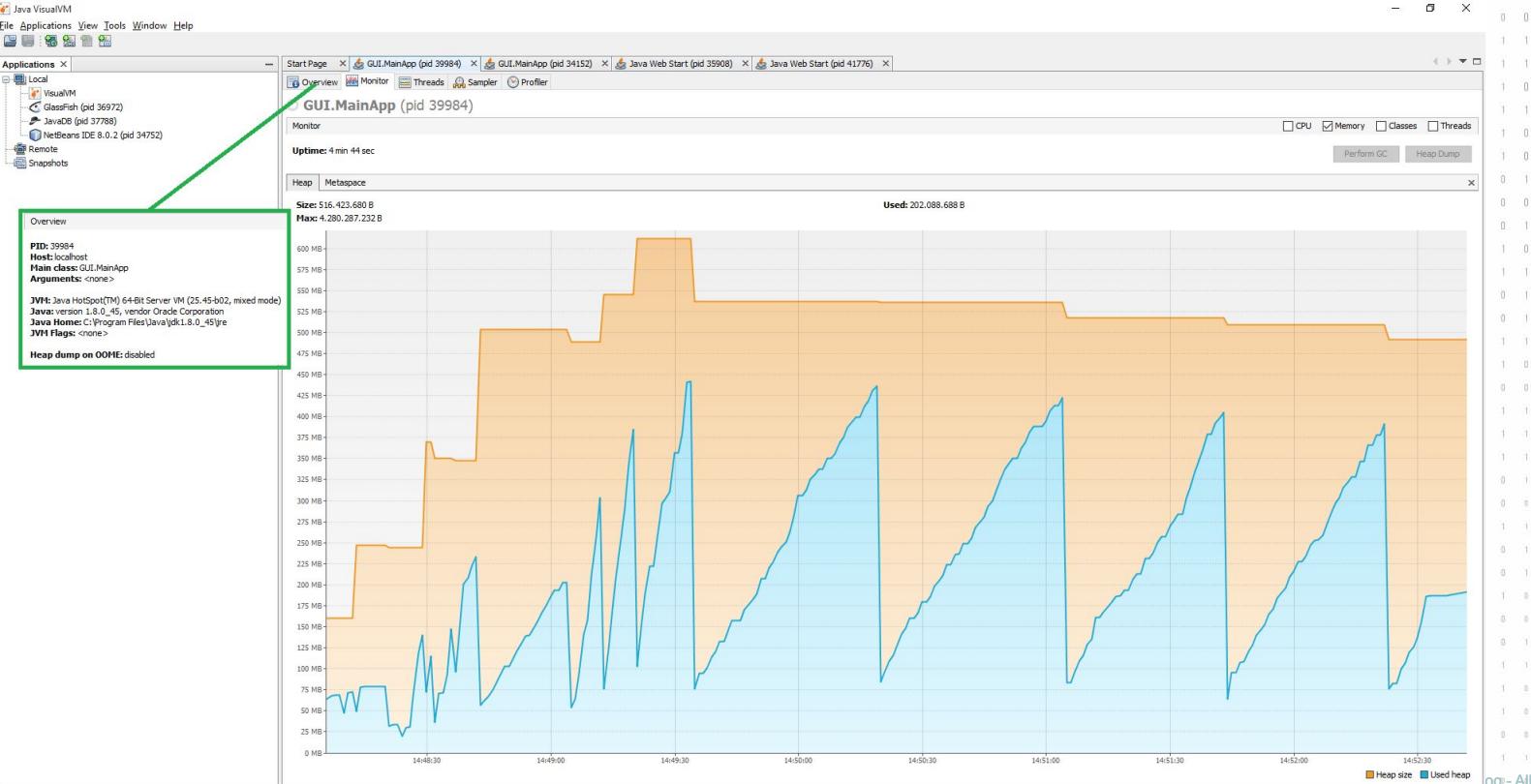
LAB5: JVM MEMORY ISSUES

- Change the Xms and Xmx JVM Heap size to simulate high load

```
helm upgrade artifactory jfrog/artifactory  
--version 7.17.1 --set artifactory.javaOpts.xms="256m"  
--set artifactory.javaOpts.xmx="512m"  
--set artifactory.javaOpts.other="-javaagent:glowroot.jar"
```

- Your Artifactory server will now have a slow response time and the UI will be slow to load
- Lets go to Glowroot and check the JVM state
- See [Artifactory JMX Mbeans](#) and add them record them in Glowroot

The JVM has a *heap* that is the runtime data area from which memory for all class instances and arrays are allocated



JVM - OUT OF MEMORY





EXCESSIVE GARBAGE COLLECTION



artifactory :9009 (pid 10737)

Monitor

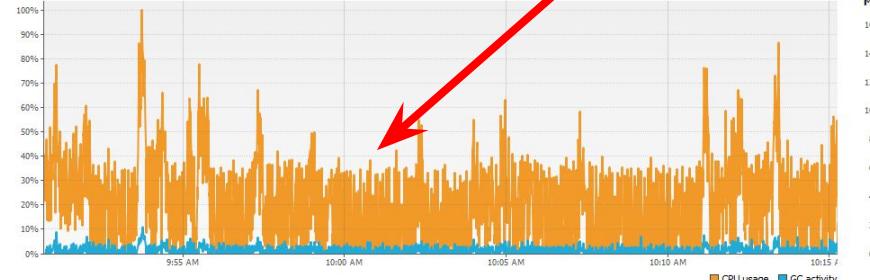
Uptime: 30 min 21 sec

CPU Memory Classes Threads

Perform GC Heap Dump

CPU

CPU usage: 2.4%



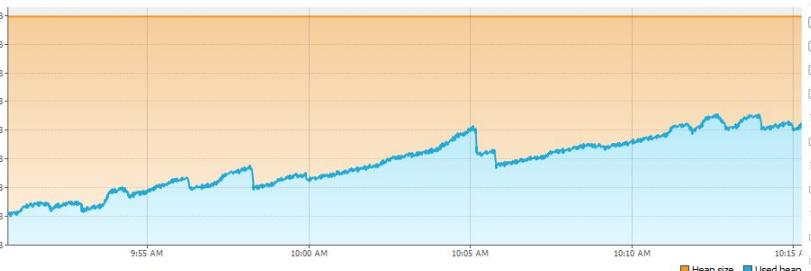
time spent on GC

x | Heap | PermGen

Size: 17,145,004,032 B

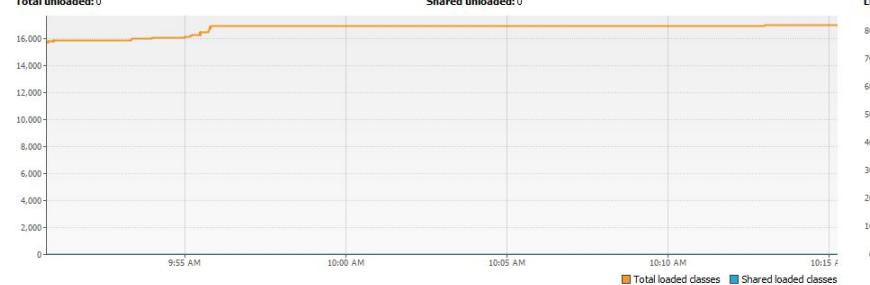
Max: 17,145,004,032 B

Used: 9,062,668,504 B



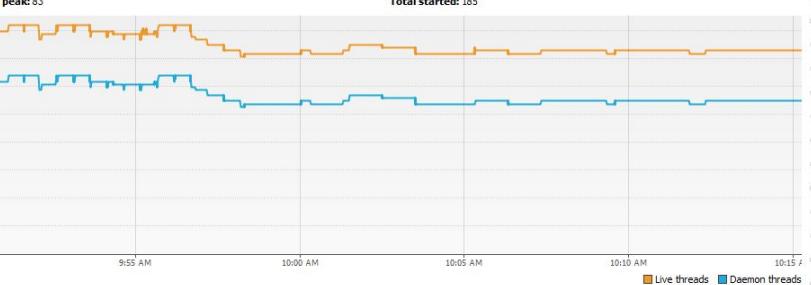
Classes

Total loaded: 17,019
Total unloaded: 0



Live: 73
Live peak: 83

Daemon: 55
Total started: 185



HOW DOES IT AFFECT PERFORMANCE?

- Java heap free space is getting lower and lower
- Unable to find a large-enough consecutive section of free memory to be allocated
- At that point, the JVM determines that a garbage collection needs to happen and it notifies the garbage collector



JVM TUNING PARAMETERS

Parameter	Description	Recommendations
-Xmx	Maximum heap size	Should be big enough (e.g. 4 Gb)
-Xms	Initial heap size	Should be $\frac{1}{4}$ to $\frac{1}{2}$ of Xmx
-Xss	Stack size	Can be small (e.g. 128 Kb)
-XX:+UseG1GC	GC to use	Use Garbage One collector

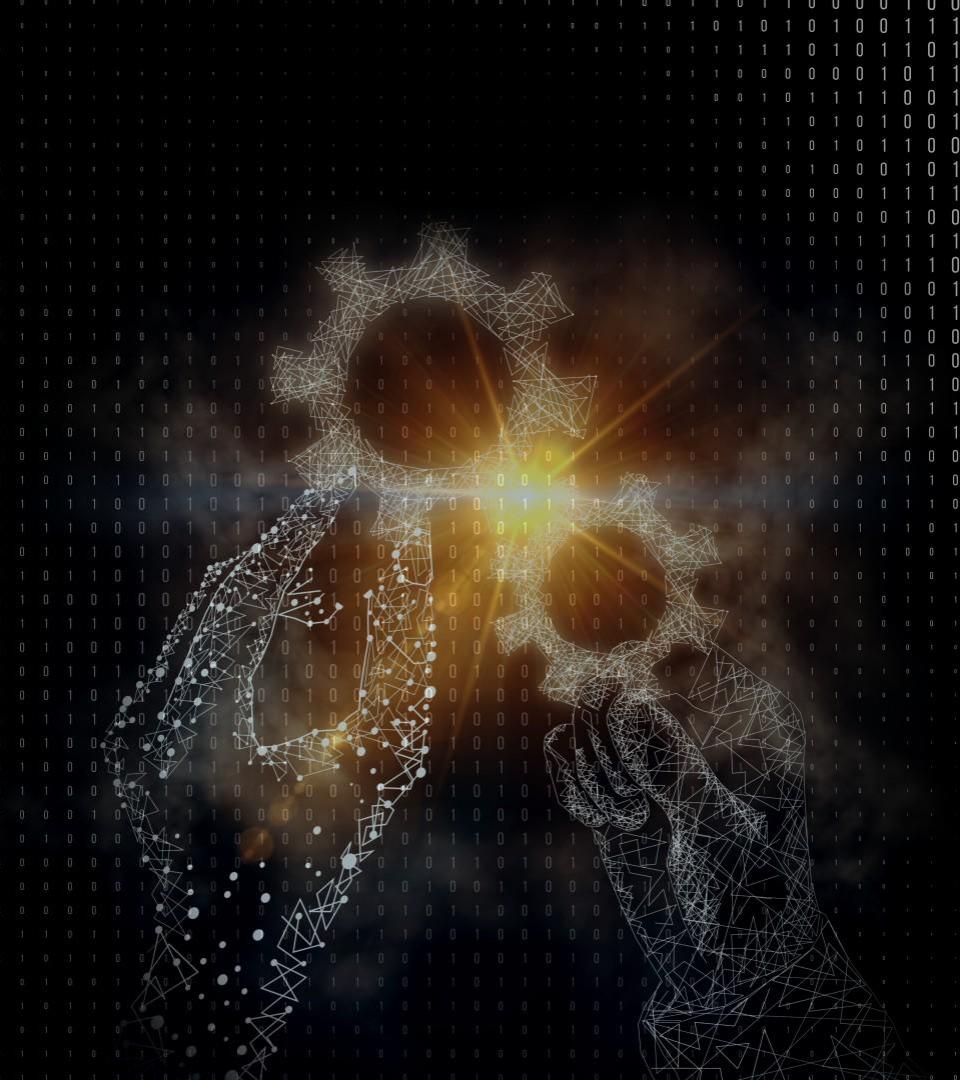


JFrog Certification

[Certification FAQs](#)

[Study Guide](#)





FAULT - LOG FILES

Appender	Level	File Name	Message Pattern
CONSOLE	warn	ARTIFACTORY_HOME/logs/catalina/catalina.out	%date \${artifactory.contextId}[%thread] [%-5p] \(%-20c{3}:%L\)- %m%n
FILE	warn	ARTIFACTORY_HOME/logs/artifactory.%i.log	%date [%thread] [%-5p] \(%-20c{3}:%L\)- %m%n
ACCESS	info	ARTIFACTORY_HOME/logs/access.%i.log	%d %m%n
IMPORT.EXPORT	DEBUG	ARTIFACTORY_HOME/logs/import.export.%i.log	%date [%-5level] \(%logger{32}:%line\) %message%
TRAFFIC	info	ARTIFACTORY_HOME/logs/traffic.%d{yyyyMMdd}	%message%
REQUEST	info	ARTIFACTORY_HOME/logs/request.%i.log	%message%
REQUEST_TRACE	info	ARTIFACTORY_HOME/logs/request_trace.%i.log	%date %message%
JFROG_ACCESS	info	ARTIFACTORY_HOME/access/logs/jfrog_access.%i.log	%date [%thread] [%-5p] \(%-20c{3}:%L\)- %m%n
JFROG_ACCESS_AUDIT	info	ARTIFACTORY_HOME/access/logs/audit.%i.log	%date %message%
SHA256_MIGRATION	info	ARTIFACTORY_HOME/logs/sha256_migration.%i.log	%date \${artifactory.contextId}[%thread] [%-5p] \(%-20c{3}:%L\)- %m%n
PATH_CHECKSUM_MIGRATION	info	ARTIFACTORY_HOME/logs/path_checksum_migration.%i.log	%date \${artifactory.contextId}[%thread] [%-5p] \(%-20c{3}:%L\)- %m%n

FAULT - TOMCAT LOGS: ARTIFACTORY_HOME/logs/catalina/

catalina.out	<p>Stdout/Stderr stream for Tomcat's JVM. Default destination for all Tomcat log statements. May include uncaught exceptions printed by <code>java.lang.ThreadGroup.uncaughtException(..)</code> or thread dumps, if requested via a system signal.</p> <p>This is the next place to check after Artifactory logs if something is wrong.</p>
localhost.<date>.log	<p>The log of the virtual host within Tomcat that runs the web application container. Normally does not contain much of interest but would be the next place to check after <code>catalina.out</code>.</p>
catalina.<date>.log	<p>The web application container log file.</p>
host-manager.<date>.log manager.<date>.log	<p>The logs of the related web applications.</p> <ul style="list-style-type: none">• host-manager for Virtual hosts• manager for the deploying and undeploying applications