IBM Cloud Private Performance and Sizing

IBM Cloud

ICP Performance Analysis

Performance is analyzed based upon 3 representative application architectures

General performance analysis

Sizing your Deployment

Tuning your Cluster

ICP Performance Analysis

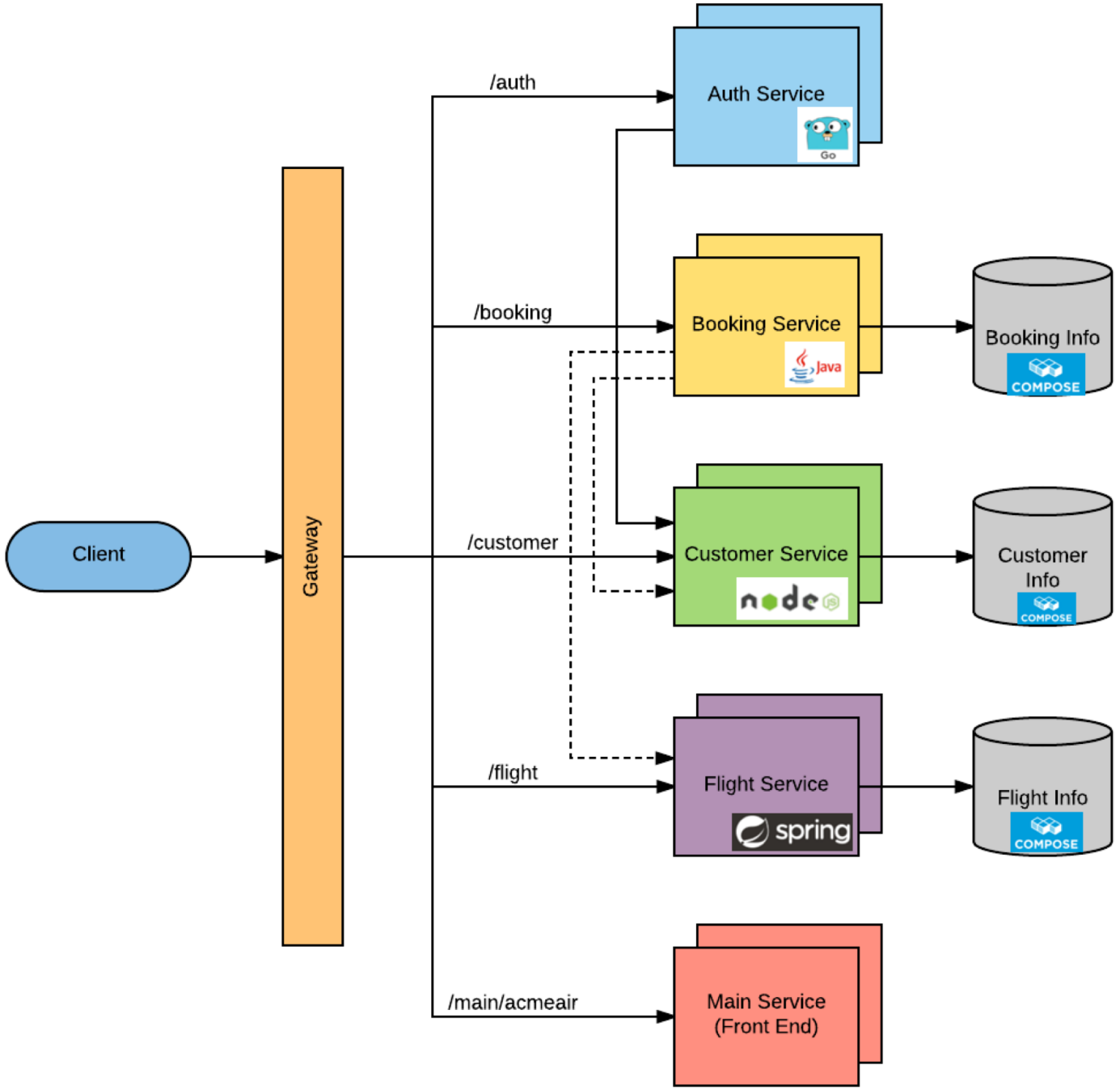
Analyzing Performance with Applications

The Polyglot

We will consider a Polyglot application referred to as Acme Air.

This application utilizes multiple programming frameworks.

The intent is to use a general case to evaluate performance and scalability of IBM Cloud Private.



Online Banking

A second more complicated application is used to simulate Online Banking

Simulation of retail online banking (UI, security and services)

Traffic is encrypted and users are authenticated

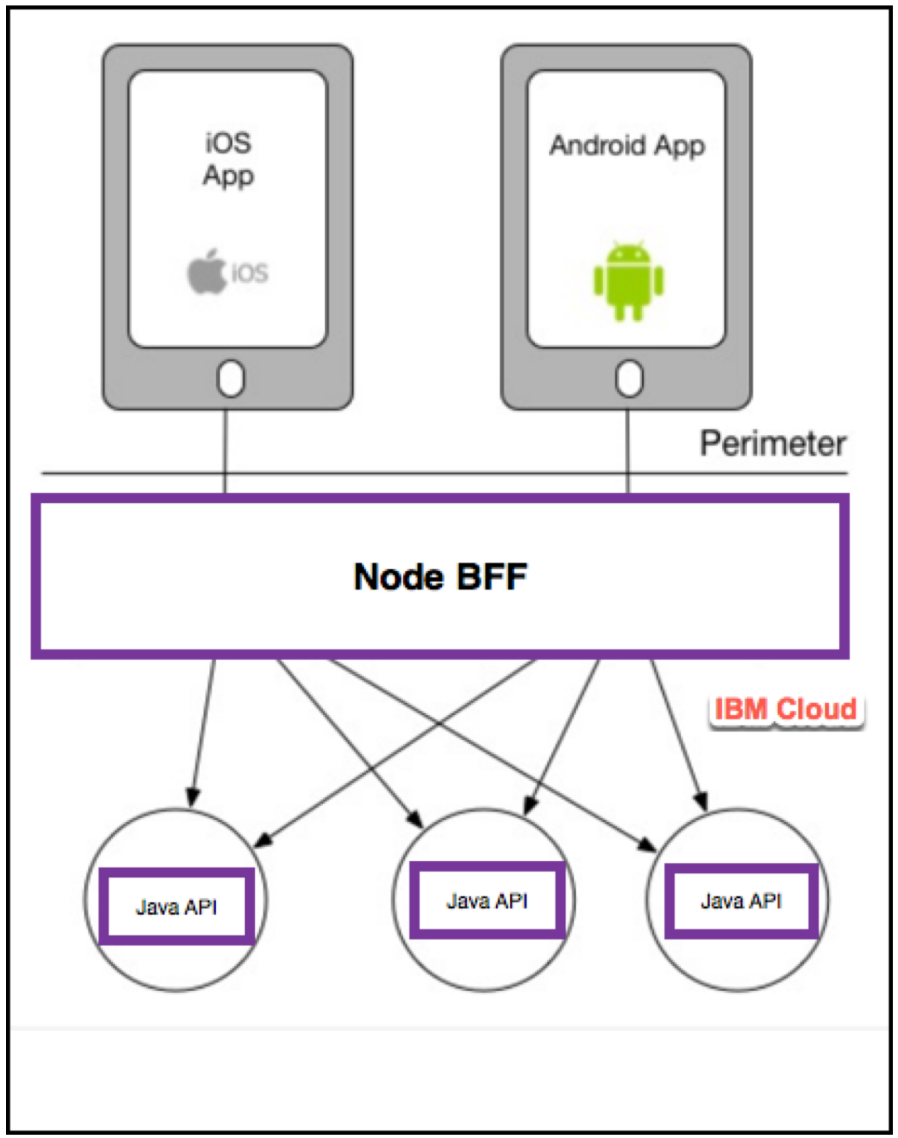
Backend services interaction has been simulated via Stub Application

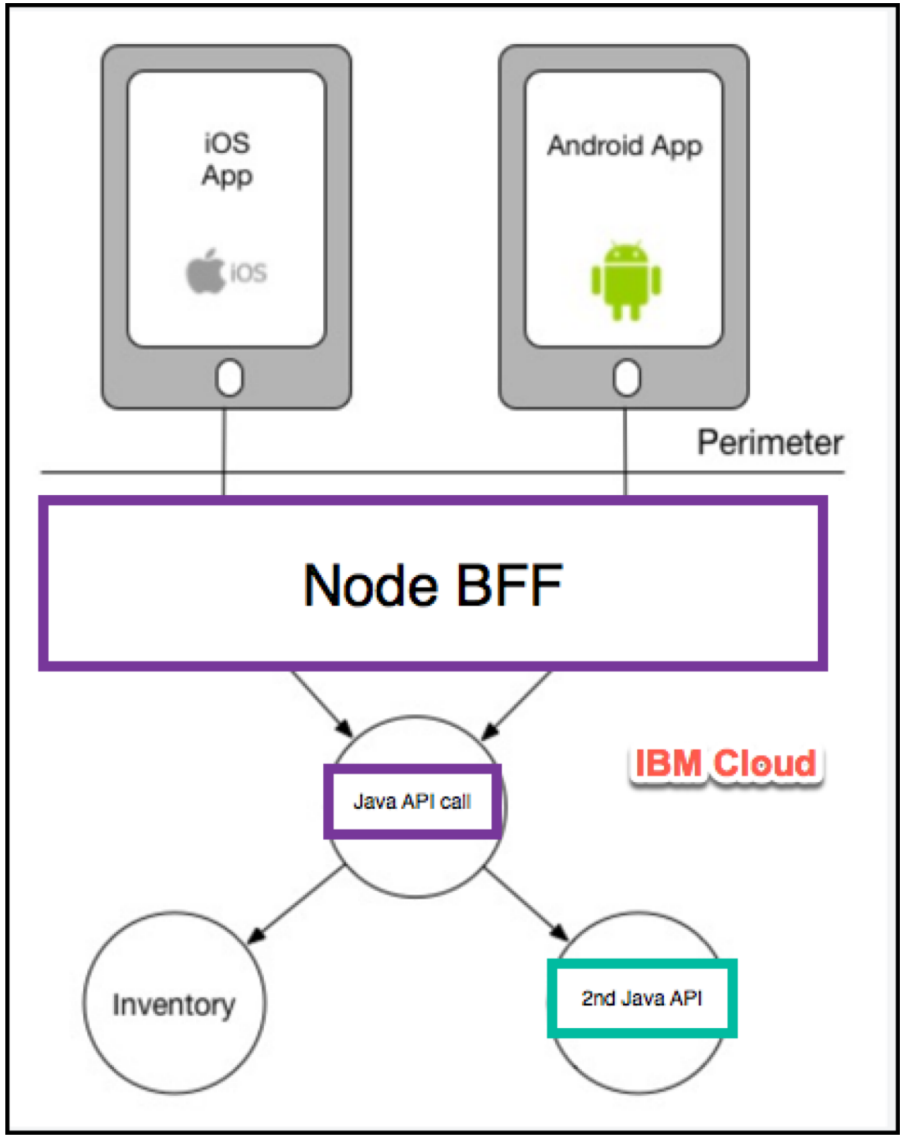
Account Summary Page is developed using Angular JS

Healthcare Microservices

Typical Microservices application

Architecture comparing 2 and 3 levels



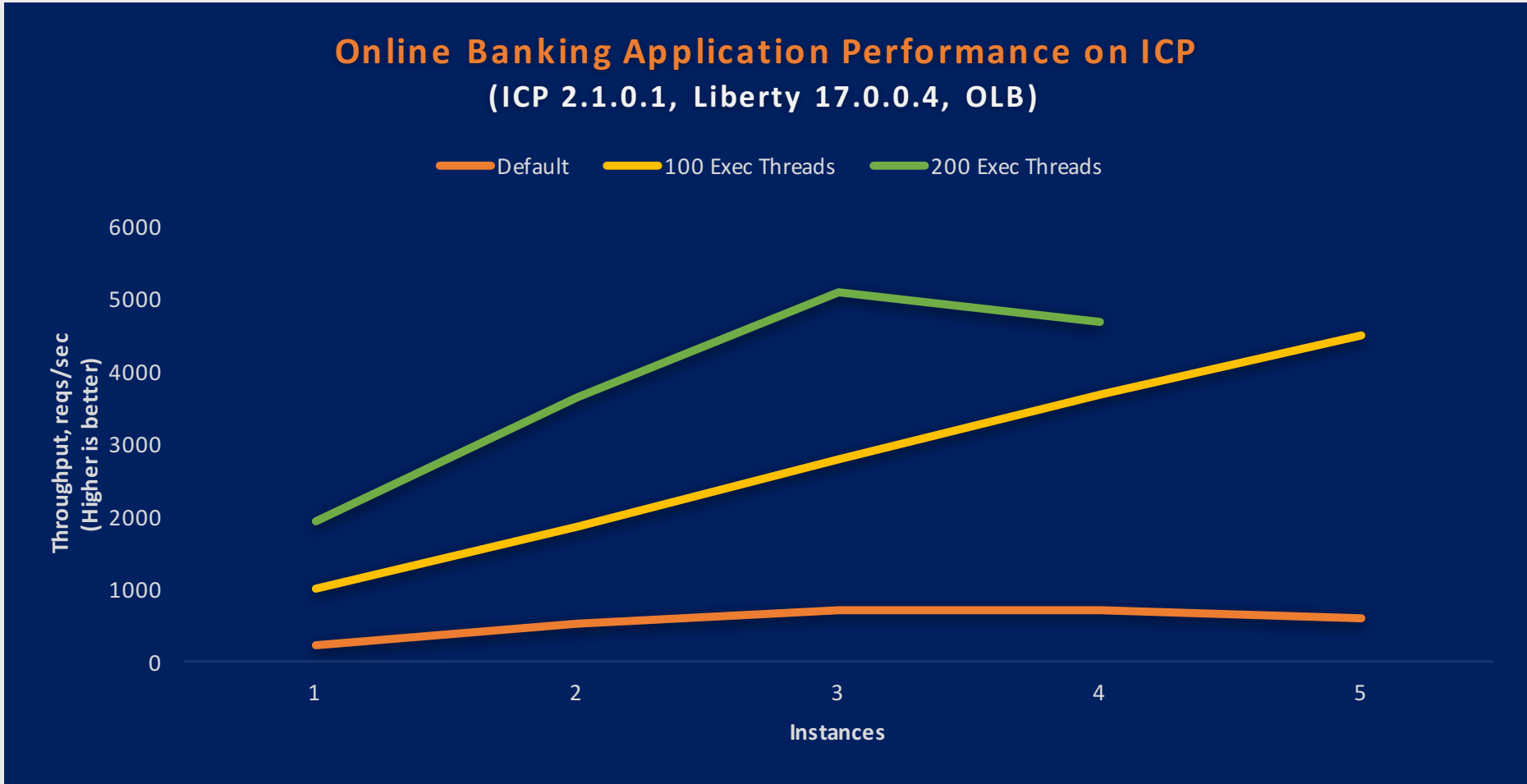


Online Banking Performance on ICP

This type of Java microservice application scales nearly linearly
Scaling this application, you have massive parallel workload coming in
The runtime (in this case Liberty) must have enough executable threads to manage
Cannot get full use of ICP resources without tuning
Able to saturate by increasing executable threads

The default (orange) line shows only using horizontal scaling does not fully leverage the cluster resources

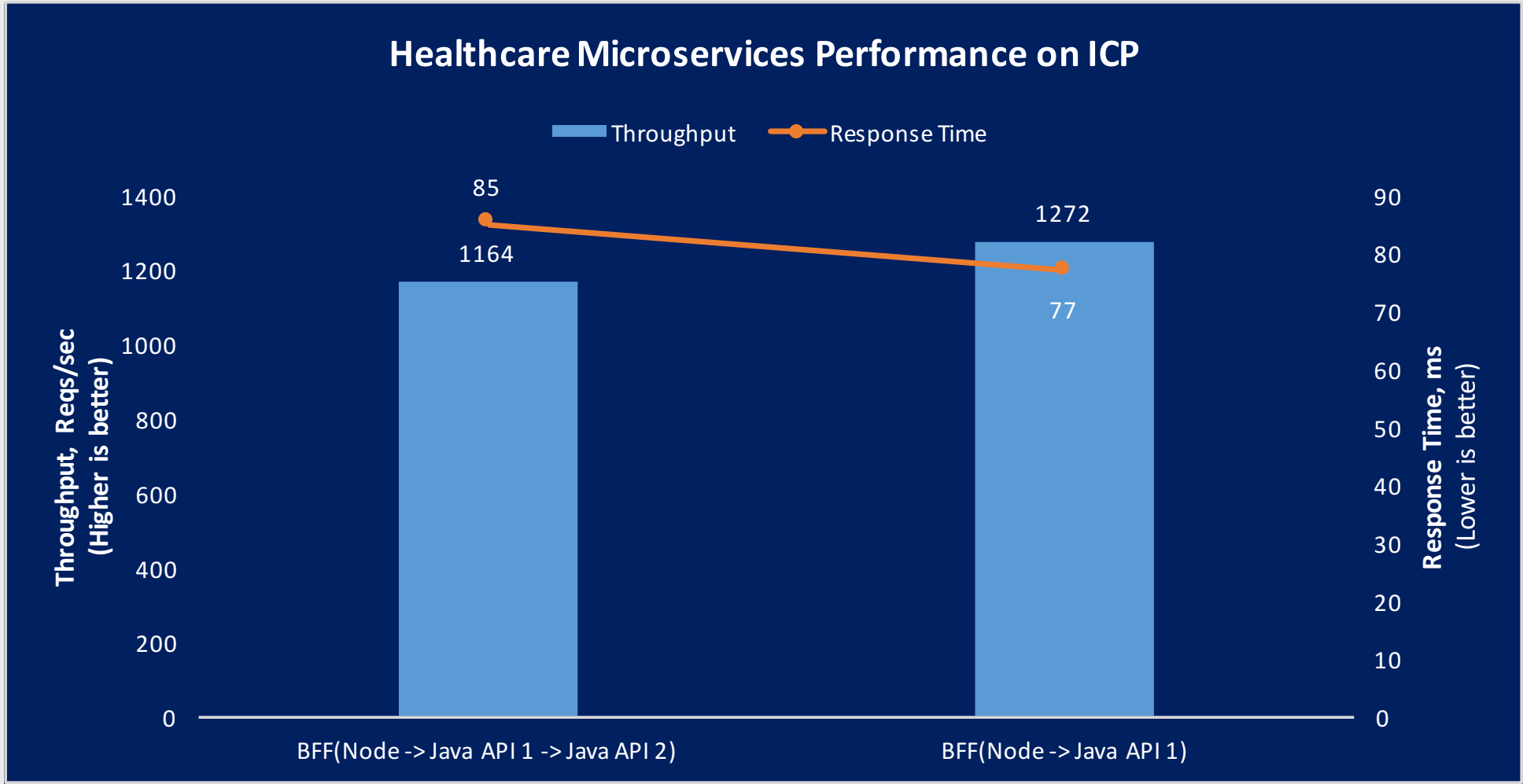
Understanding the application and its run time shows increasing the Liberty executor thread pool size is crucial for this Java application



Healthcare Application Performance on ICP

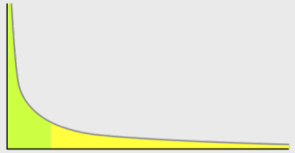
Comparison of 2 and 3 level BFF service calls
We desire 3 level BFF microservices
Many platforms suffer massive latency when making the jump from 2 level to 3 level

The results of this testing shows only an approximate 10% difference when moving from 2 to 3 layers on ICP



Healthcare Application Performance on ICP Long-Tail Latency

Statistically speaking, the long tail of distributions is the portion of the distribution having a large number of instances or occurrences far from the head (mode or median) of the distribution.

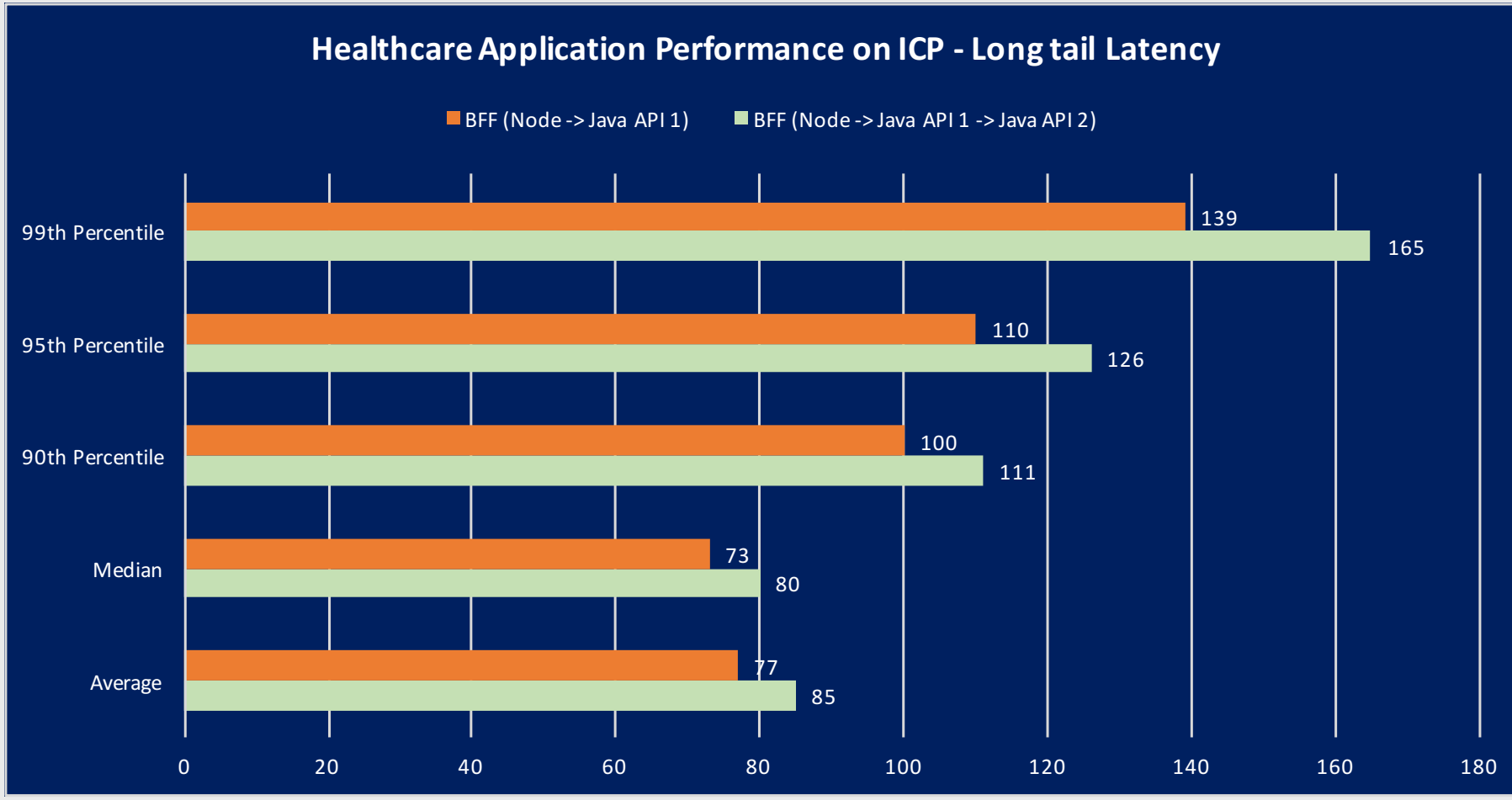


We can consider long-tail latency manifested in the difference between the 90 and 99 percentiles
Long-Tail latency being high typically results in a negative user experience

It can be a difficult problem to solve

Not uncommon in some instances to see 10X (1000%) difference in this region of a distribution

The test shown here shows 0.45X (45%) for IBM Cloud Private



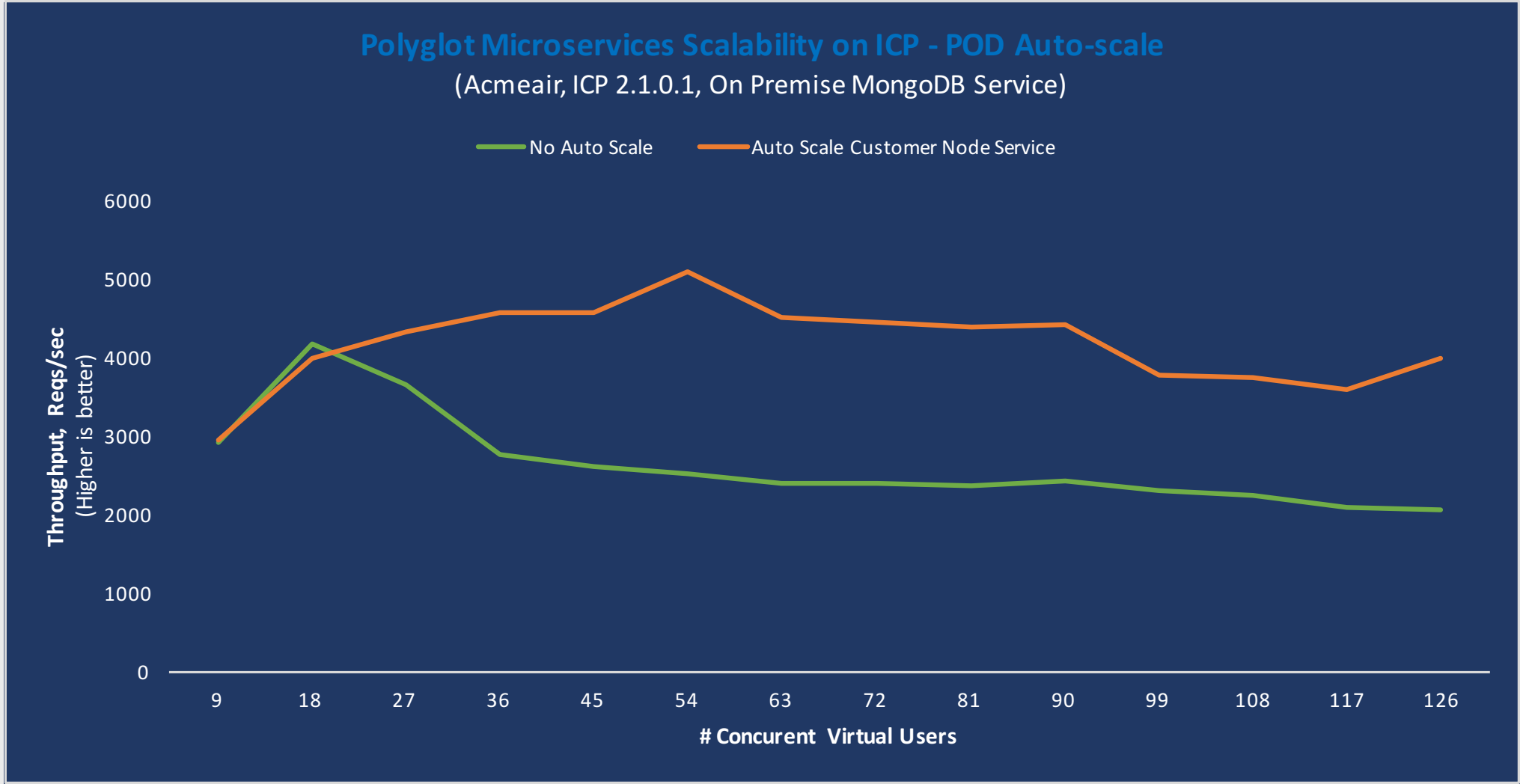
Polyglot Microservices Performance and Scalability on ICP

The test being run in this example is a node.js application accessing a MongoDB data service
We are comparing ”No” and ”Auto” scaling on the node.js service

The services here are deployed within the same cluster

The inability of the application to scale (green line) shows the overall degradation that can occur

These principles still apply as we consider locating the MongoDB externally to the cluster



Importance of POD Auto-Scaling on Application Performance

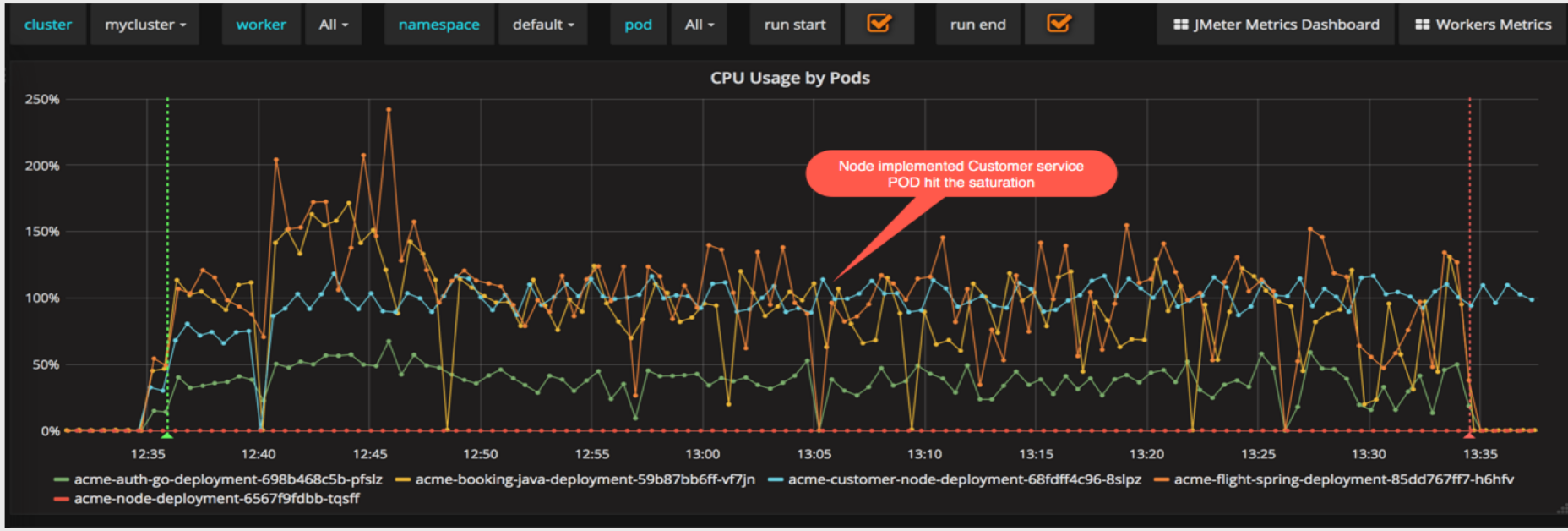
The top chart shows no pod auto-scaling and the bottom chart is with auto-scaling

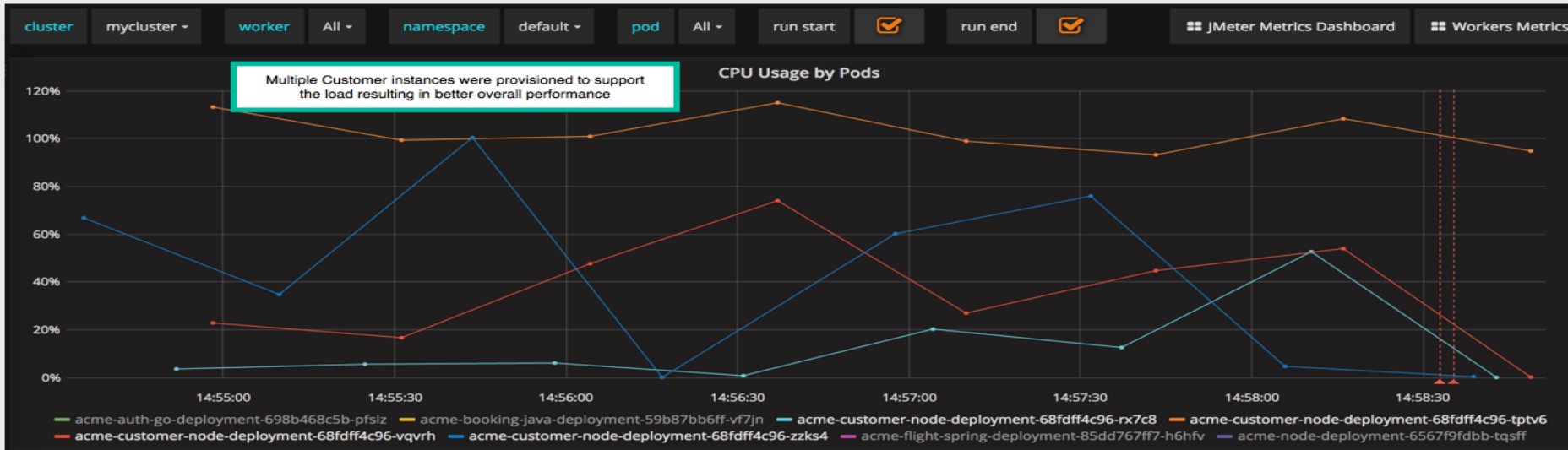
Top Chart: Blue line shows a pod where the customer service is deployed

Top Chart: Performance is for the most part pegged and demonstrates typical node.js behavior

Top Chart: Unable to push beyond ~100 threads

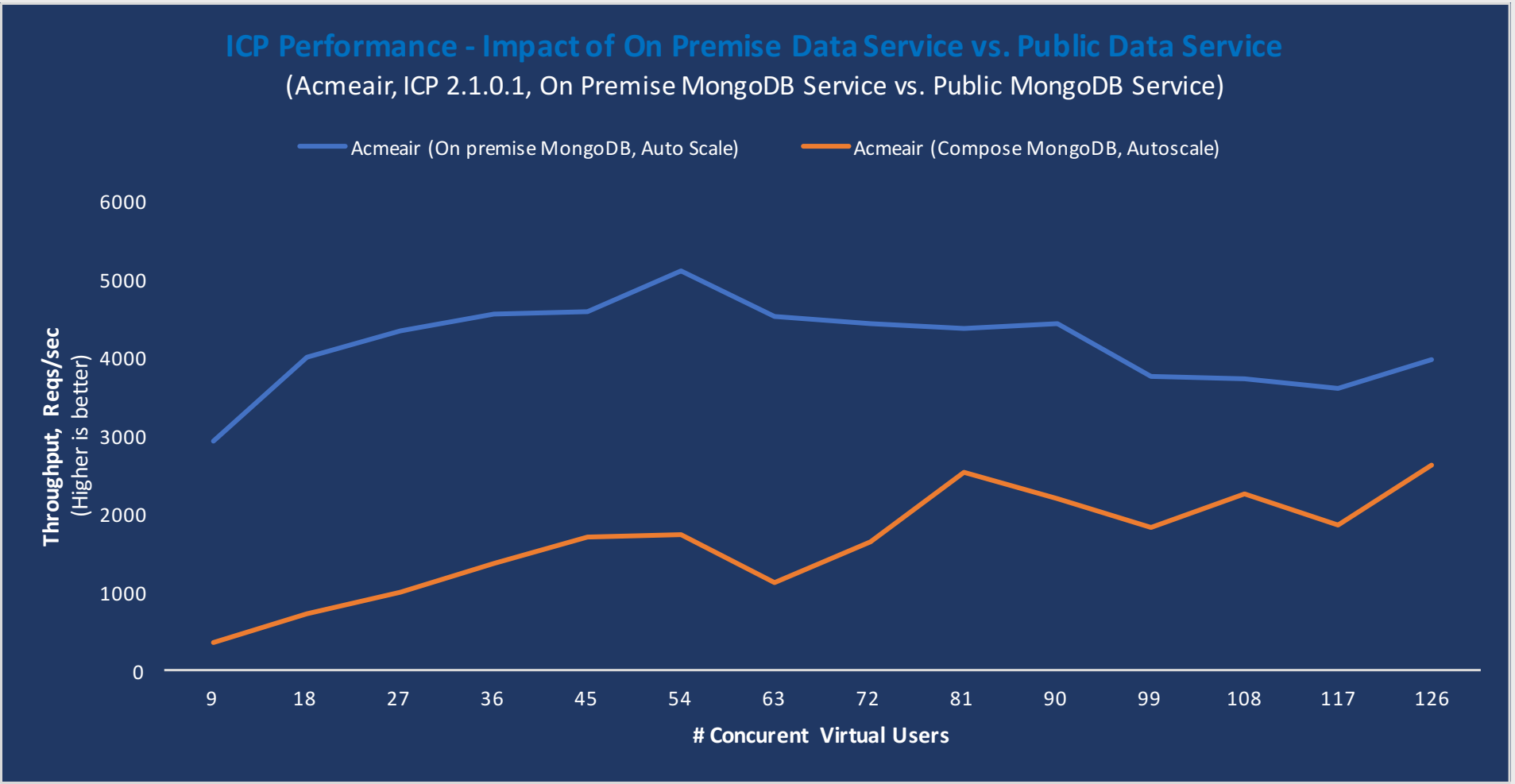
Bottom Chart: Much better performance as the app scales to match demand





Compare On Premise v. Public Data Service

Both workloads are shown using auto-scaling

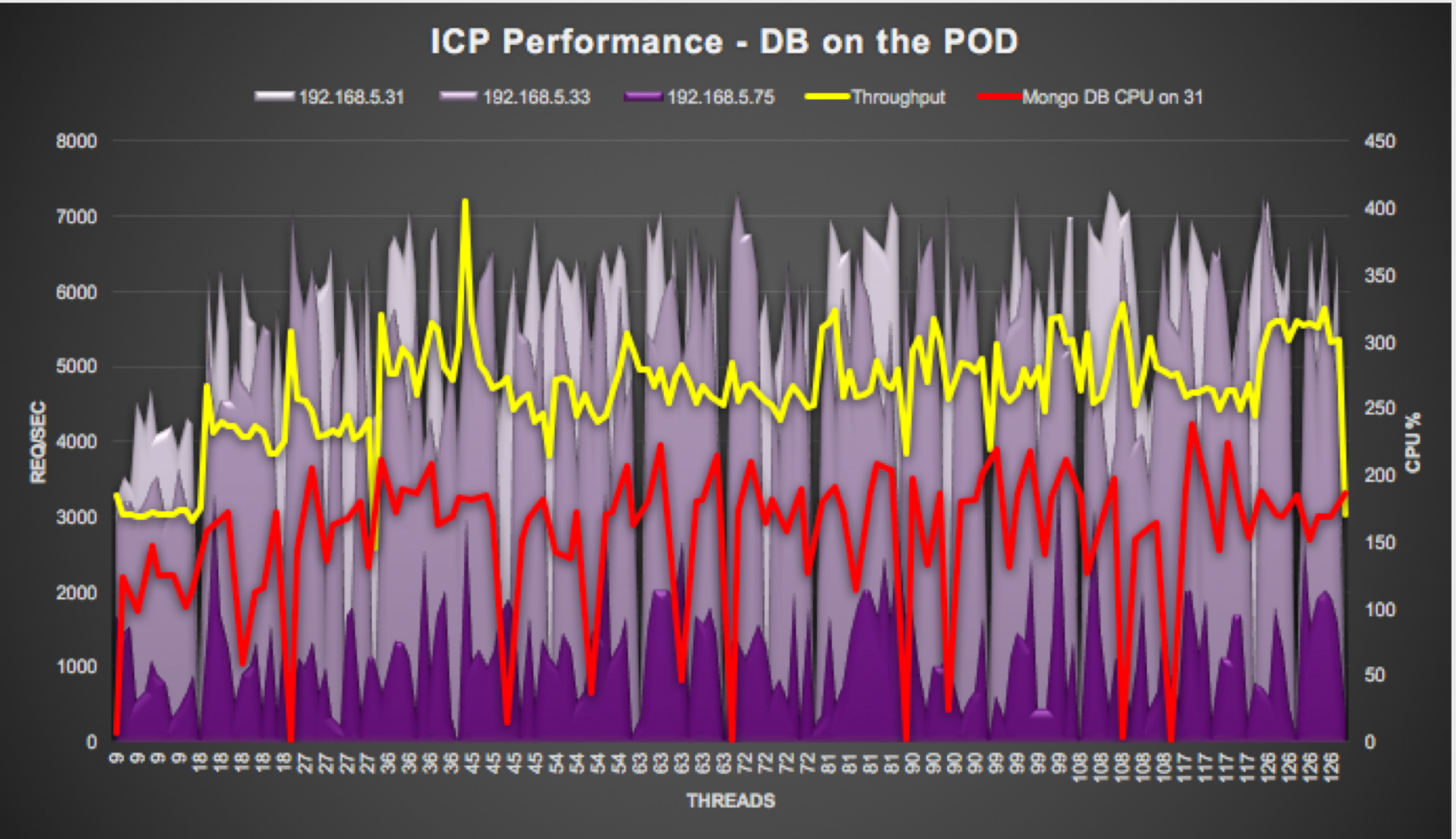


Shows the impact of network latency introduced with the public data service (Compose MongoDB)

We see linear scaling in the public scenario with lower performance throughout as compared to the on premise configuration

ICP Performance: MongoDB on ICP Cluster

Consider polyglot microservices with MongoDB deployed via Helm locally into the same cluster
Placement of the DB is crucial for scalability of the overall application



The Yellow line shows the throughput of the polyglot application as load is increased

The Red line shows the associated MongoDB pod CPU activity

The remaining (purple) represents the 3 worker nodes in this cluster (saturates at ~400% CPU)

Maximum performance would occur by separating the workload for MongoDB

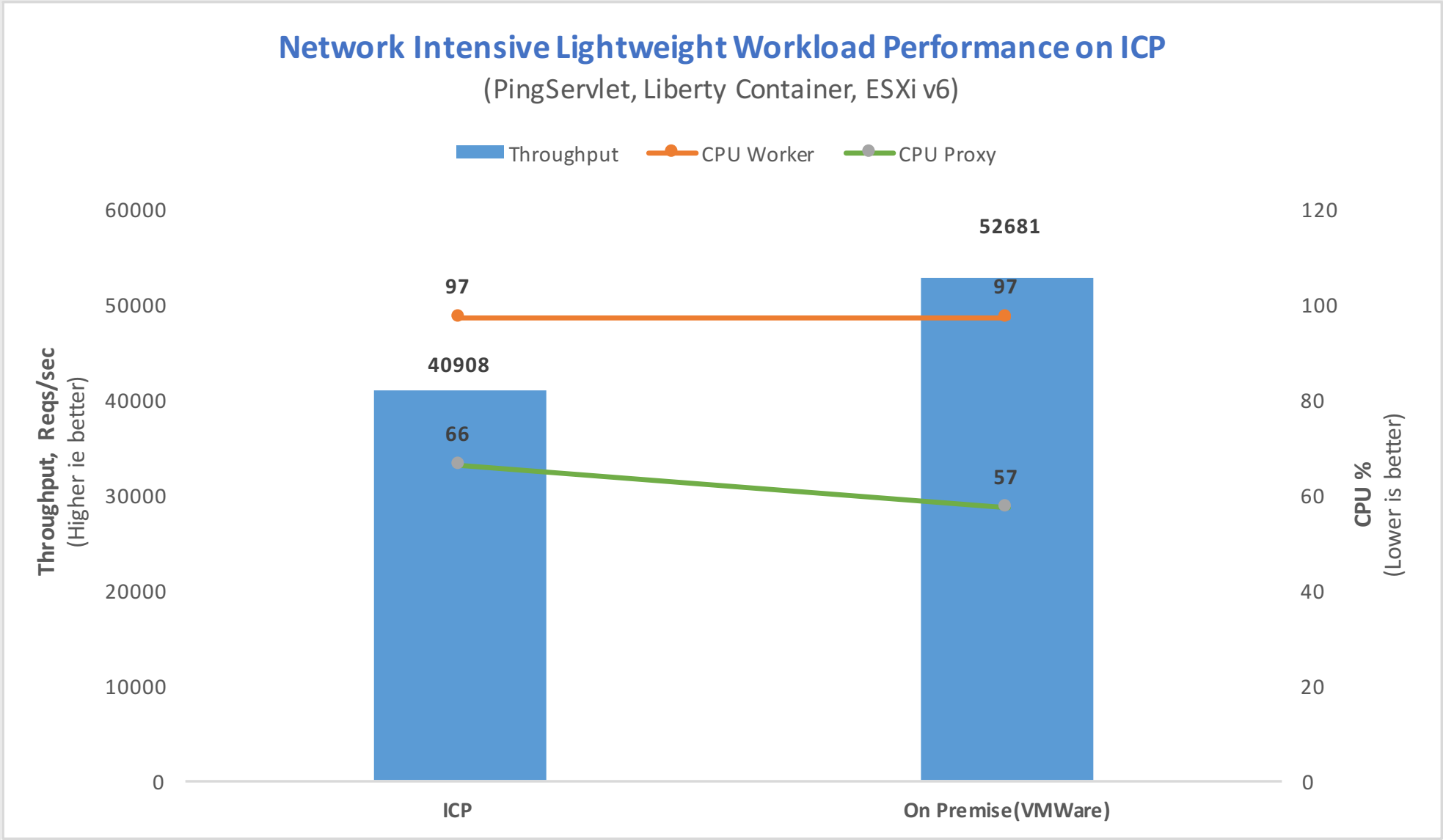
Network Intensive Application Performance on ICP

Consider the moving of a legacy application that is network intensive to IBM Cloud Private

For network intensive virtualized workloads migrating to ICP we may require 1/3 more compute to support the same transaction load (as network resources transition to CPU)

Docker networking overhead is the source for this load

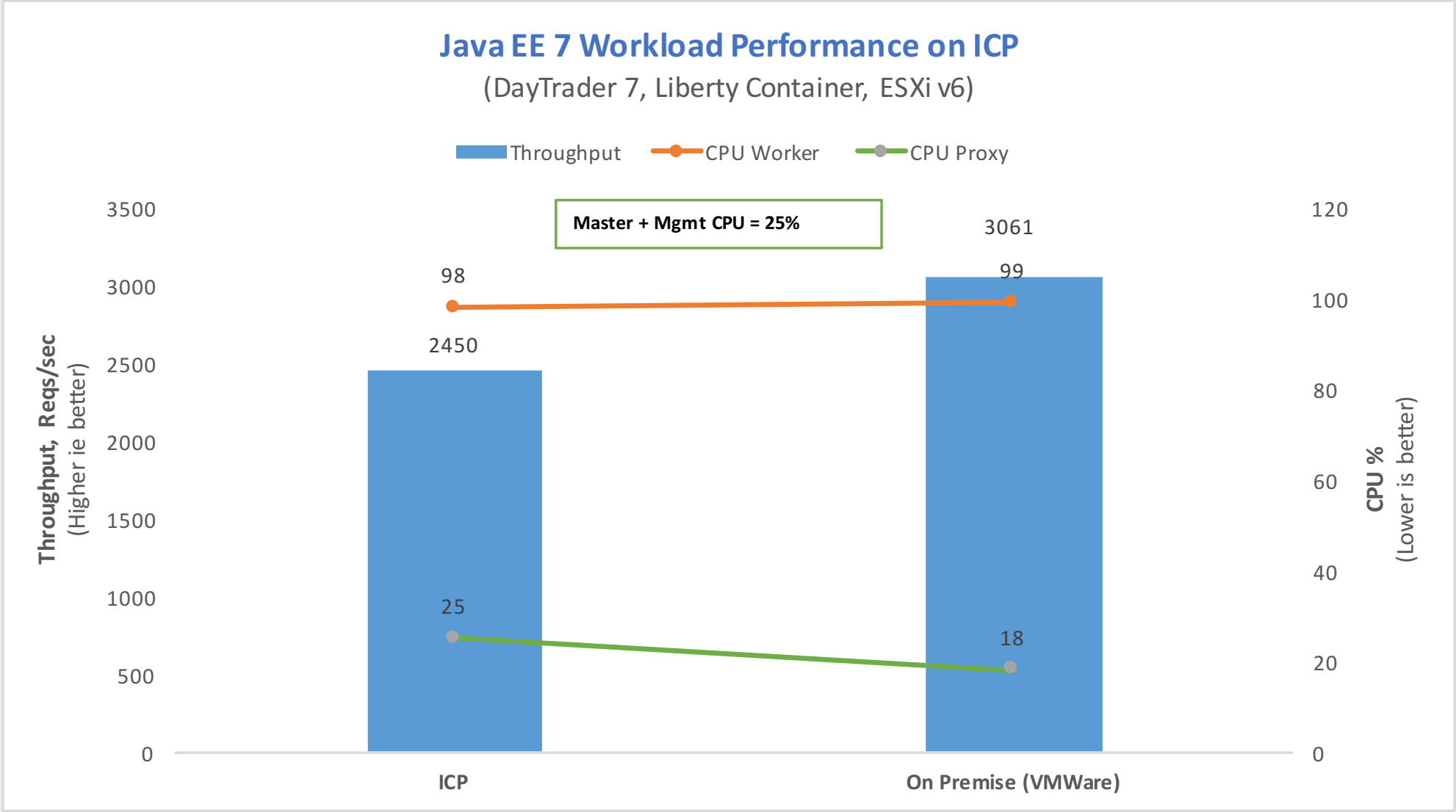
Add an additional 25% capacity for Management / Master nodes and 30% for the running the application



Java EE Application Performance on ICP

For a Java EE 7 workload migrating from on premise virtualized environment to ICP we require an additional 25% capacity to support similar transaction load (this includes the control plane masters, management, etc.)

This is once again due mostly to Docker networking overhead

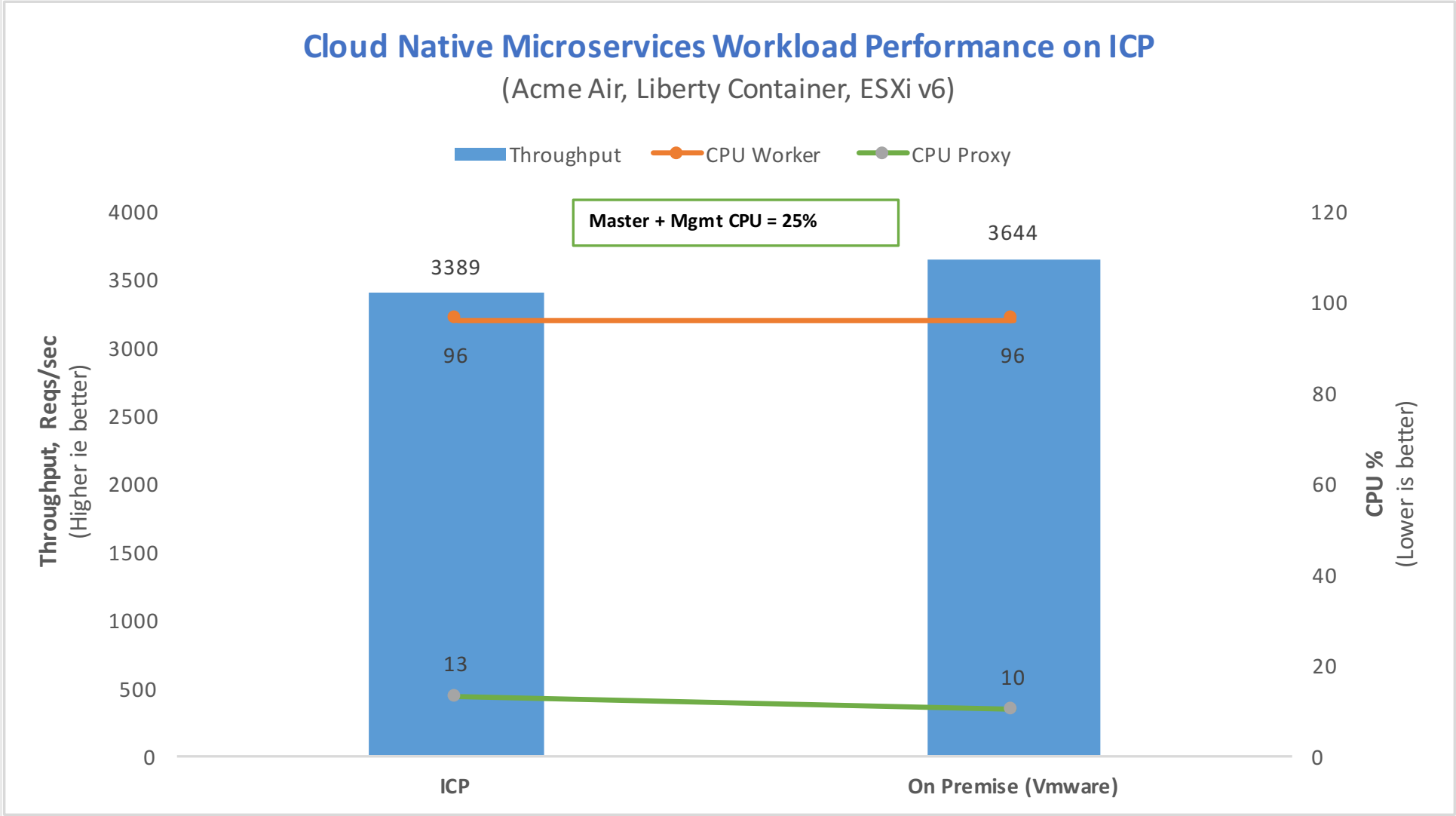


Cloud Native Application Performance on ICP

For cloud native microservices migrating to ICP from virtualized infrastructure require ~7% additional capacity

Does not include the cluster overhead of ~25% infrastructure increase for the control plane (Master / Management)

Due mostly to Docker networking overhead



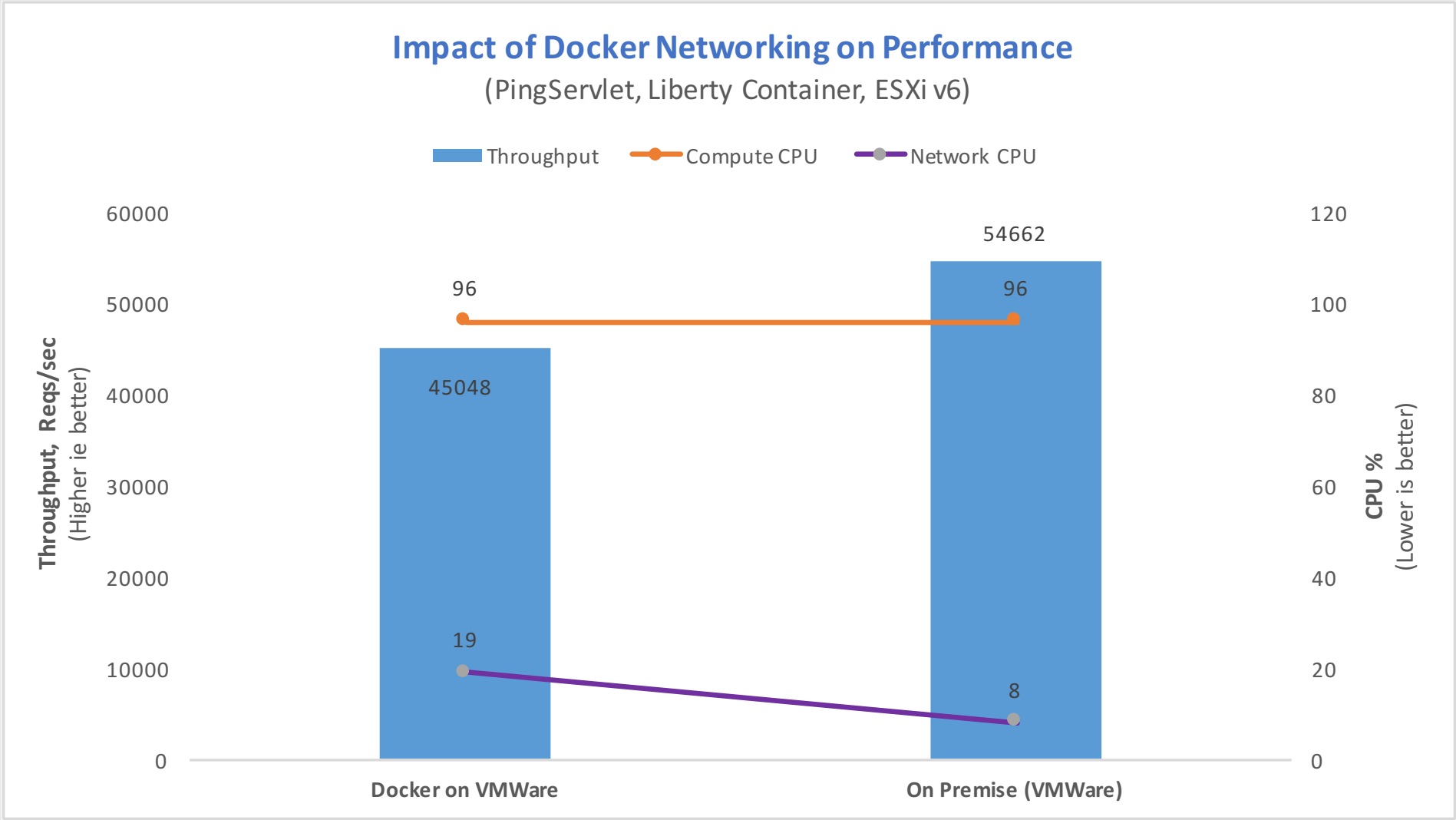
Reason for Additional Overhead on ICP

Compare Docker on virtualized infrastructure with virtualized alone

We see overhead from Docker networking

Inefficient handling of software and networking transaction are manifested in higher CPU utilization

This behavior is currently being worked from the platform perspective



Sizing Your Deployment

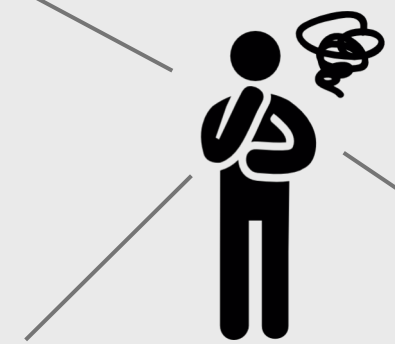
Choosing a Topology

Considering Capacity

To accurately size you ICP cluster you must consider your workload

What type of applications are you running? Are they microservices … which frameworks / runtimes … ?

Are you running any large containers that require large compute / memory footprints?



What are the current resource requirements to run your workloads and how are these requirements expected to change?

Do your applications require persistent or external storage? What type of performance is required for this storage? How many instances of persistent volumes of each type are required?

The Sandbox

Assumptions

- Resiliency is a secondary consideration
- Maximum pod size is small
- Small amount of workload
- Vulnerability is not a concern

Role	Count	vCPU / Core	Memory (GB)	Storage (GB)
Master, Boot, Management, etcd	1	4	8	250
Proxy	1	2	4	140
Worker	3	4	8	200
NFS	1	2	4	300

Production Deployment

Assumptions

- This is one possible starting point
- Storage must be sized as its own exercise
- Etcd should be broken out when worker nodes surpass 50
- The number and specific size of worker nodes depends heavily on the personality of your workload

Role	Count	vCPU / Core	Memory (GB)	Storage (GB)
Boot	1	4	8	150
Master	3 or 5	8	32	300
Etcd	1 - 3	4	16	150
Management	2 or 3	8	32	300
Proxy	3	4	16	250
Vulnerability Advisor	1 - 3	8	32	500
Worker	5+	8	32	250

Since sizing best practices change regularly with versions and updates, please see this article for the most current shirt size guidelines https://github.com/ibm-cloud-architecture/refarch-privatecloud/blob/master/Sizing.md

Boot Node

Externalizing your boot node alleviates some resource constraints during cluster installation

Boot Node Considerations

- Externalize for deployment of cluster convenience

Master Node

Only a single Master Node is active at any time

Master Node Considerations

- At least 3 nodes ensures a quorum can be reached upon failure
- Additional nodes help ensure the ability to recovery from failure
- Consider minimum recommendations to be applicable to only non-production environments
- Can Load Balance masters, but only one is the primary master

etcd Node

Very large clusters will require externalizing etcd to its own node(s)

etcd Node Considerations

- Shared with Calico
- Scale to meet a larger number of small transactions

Management Node

Your production clusters should use independent management nodes

Management Node Considerations

- Larger clusters with more workload thus require larger management nodes
- Fewer large nodes will have the same impact as many small nodes

Proxy Node

Understand your workload to tune and size appropriately

Proxy Node Considerations

- For compute sizing consider total resource sizing versus the # of nodes
- You can tune your ingress controller to your workload
- Your proxy VIP will point only to a single node at a time
- Consider optionally load balancer to spread workload to your proxy nodes

Worker Node

Consider YOUR workload to size your worker nodes

Worker Node Considerations

- If your cluster has a small amount of workload consider increasing the number of worker nodes while decreasing the size of the nodes (for adequate headspace, efficiency, mobility, resiliency)
- If you have large pods your worker nodes should be larger to accommodate workload mobility
- Java workloads typically use 4 x CPU for memory
- Other application frameworks may be closer to 2 x CPU = Memory

Tuning IBM Cloud Private

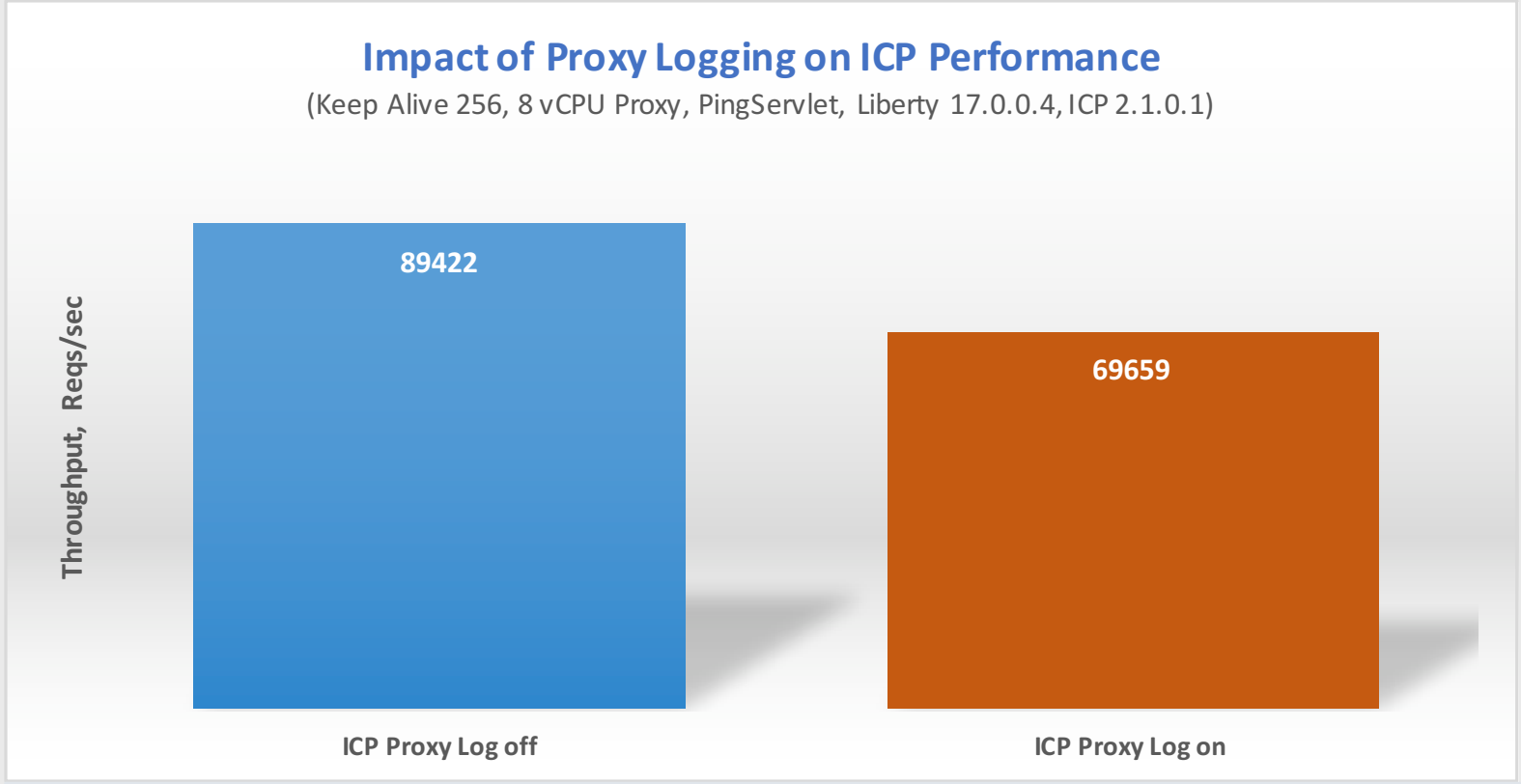
Best-Practices for Tuning the Cluster

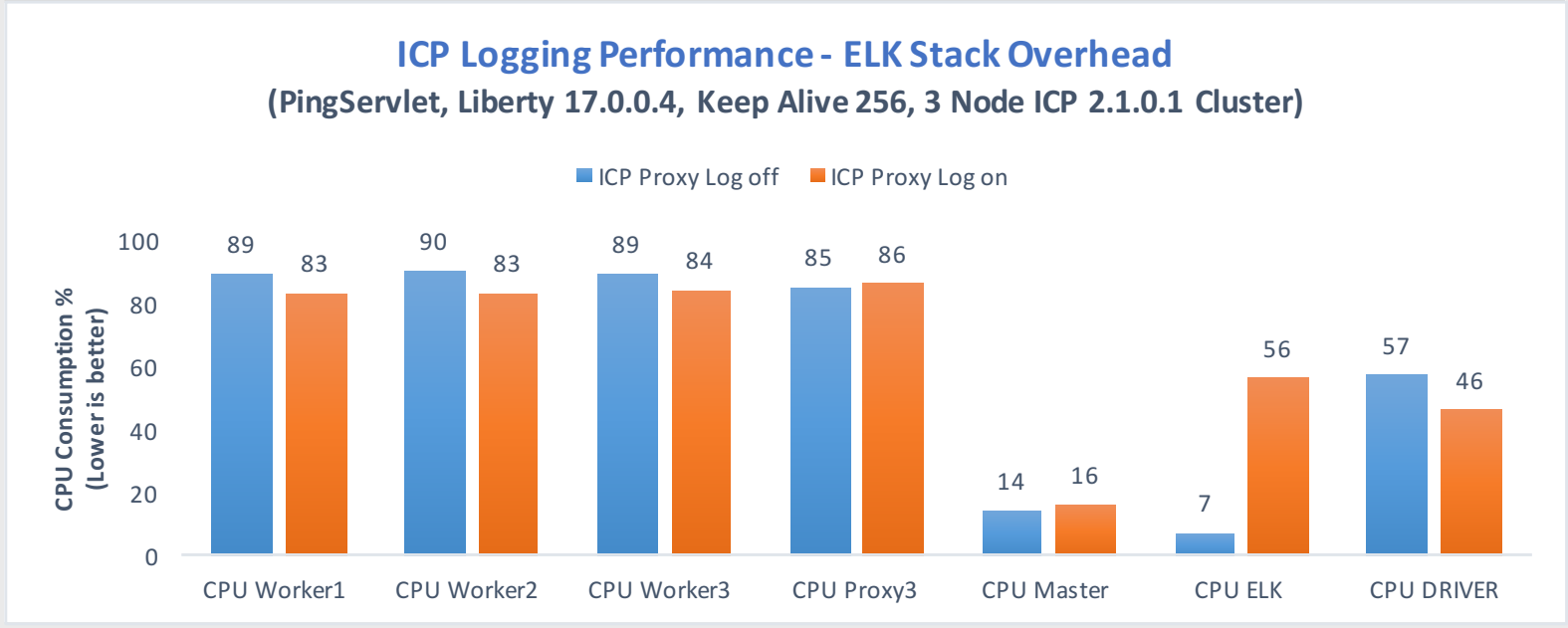
ICP Tuning: The Impact of ICP Access Logging

Performance is impacted by Proxy logging (comparing blue and orange bars in both graphs)

Network intensive loads show this can accumulate up to 30% additional overhead

The overhead is mostly attributable to ELK stack activity





ICP Application Logging

Configure log rotate for Docker (max-size, max-file)

Without log rotate disk may fill and trigger pod eviction (and image garbage collection)

Add install can be configured in config.yaml

If Docker is installed independently

- Update Docker services with “—log-opt max-size=10m –log-opt max-file=10”
- Add the config to /etc/docker/daemon.json

```
## The maximum size of the log before it is rolled
docker_log_max_size: 50m
## The maximum number of log files that can be present
docker_log_max_file: 10
```

Options

The `json-file` logging driver supports the following logging options:

Option	Description	Example value
`max-size`	The maximum size of the log before it is rolled. A positive integer plus a modifier representing the unit of measure (`k` , `m` , or `g`). Defaults to -1 (unlimited).	`--log-opt max-size=10m`
`max-file`	The maximum number of log files that can be present. If rolling the logs creates excess files, the oldest file is removed. **Only effective when `max-size` is also set.** A positive integer. Defaults to 1.	`--log-opt max-file=3`

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m"
    "max-file": "10"
  }
}
```

Impact of Proxy Size and Upstream Keep Alive

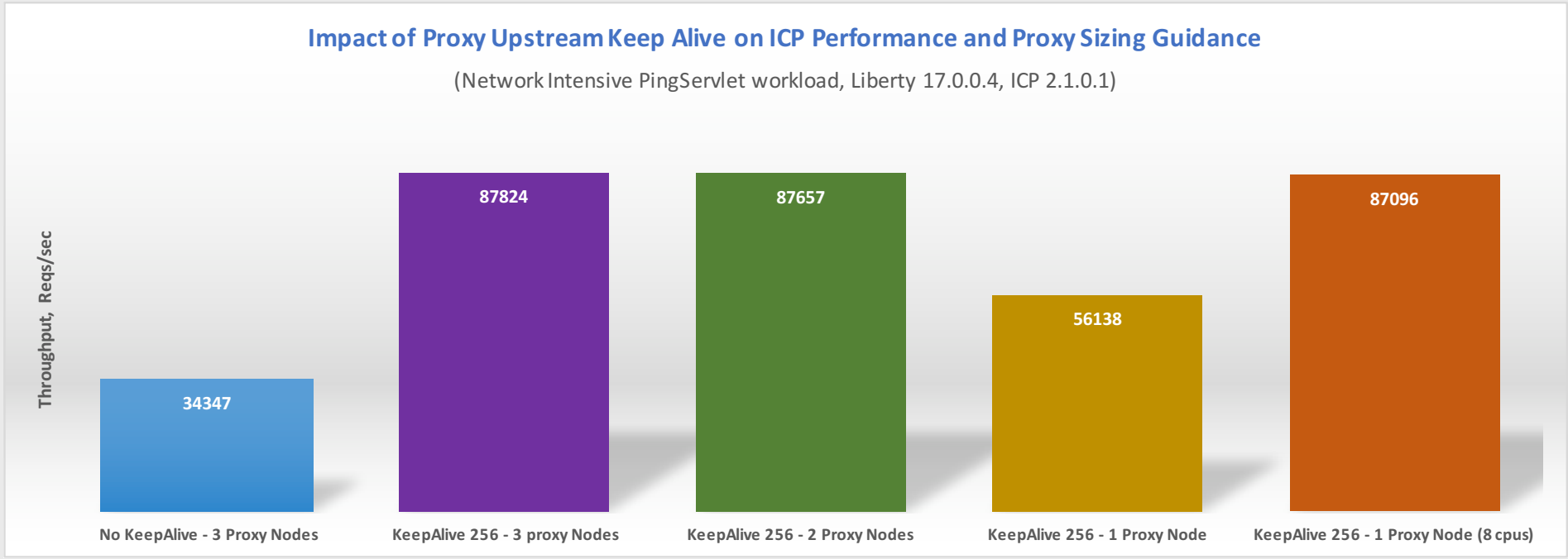
Shows the importance of upstream keep-alive

Proxy node capacity is vital, ensure yours is sufficient

For network intensive workloads, bigger proxy node can be more efficient than multiple smaller proxy nodes

Default keep-alive is 32 – consider tuning this based upon your workload

Note: Keep-alive has been enabled by default since ICP 2.1.0.2



Ingress Parameters

- Ingress parameters matter
- Check the configmap
- Update this configmap for tuning performance
- Among the settings to consider for your workload:
  - Max-worker-connections
  - Worker-processes
- Reference the NGINX / K8s documentation for detail description: https://github.com/kubernetes/ingress-nginx
- Use initContainer to adjust systctl default values: kubectl patch ds -n kube-system nginx-ingress-lb-amd64 --patch="$(cat patch.json)"

```
spec:
  containers:
  - args:
    - /nginx-ingress-controller
        --default-backend-service=$(POD_NAMESPACE)/default-http-backend
    - --configmap=$(POD_NAMESPACE)/nginx-load-balancer-conf
```

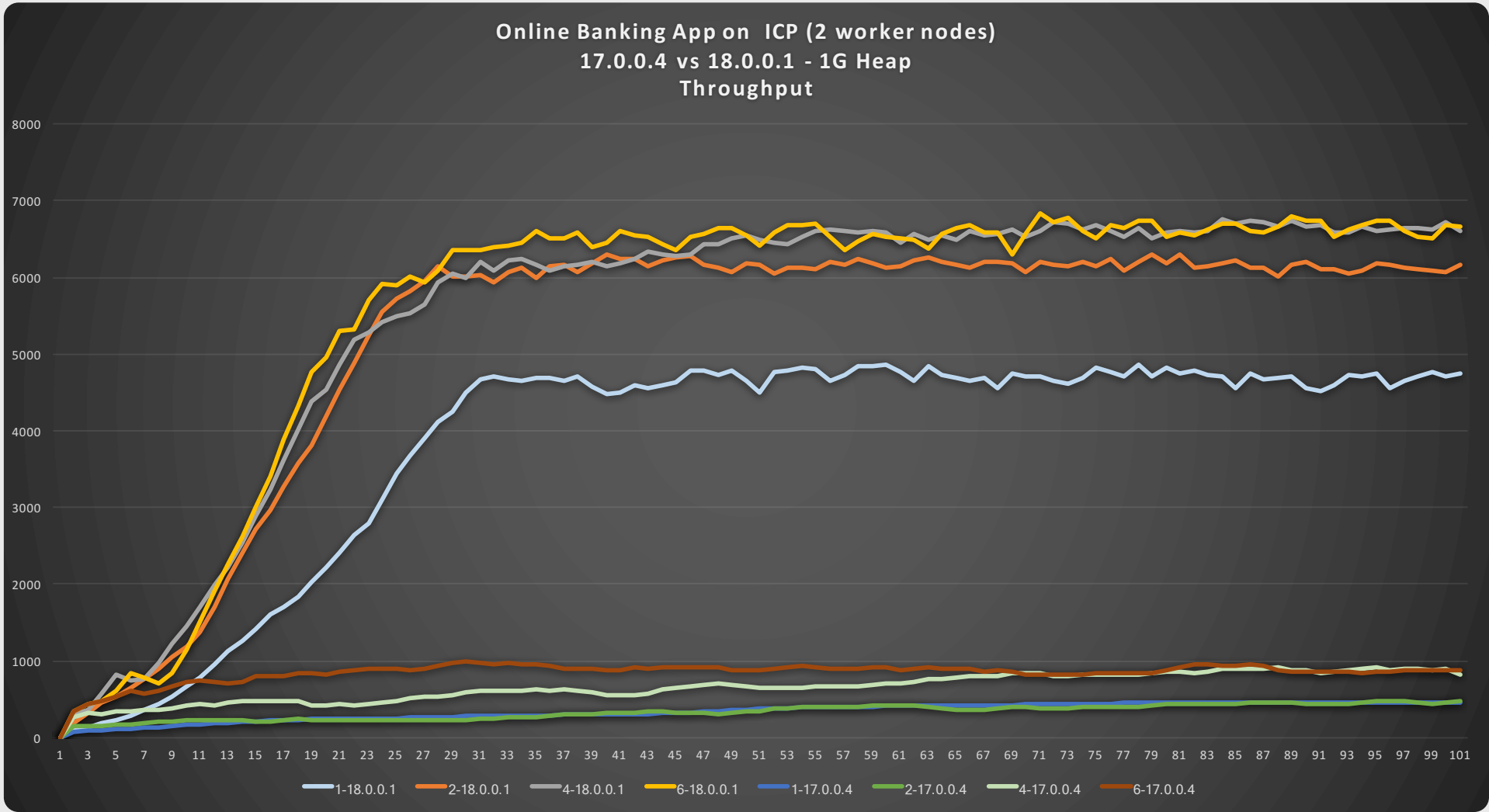
```
root@gyliu-ubuntu-1:/etc/cfc/pods# kubectl get cm -n kube-system nginx-load-balancer-conf -oyaml
apiVersion: v1
data:
  body-size: "0"
  disable-access-log: "true"
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"body-size":"0","disable-access-log":"true"},"kind":"ConfigMap"
:"kube-system"}}
  creationTimestamp: 2018-02-23T11:22:31Z
  name: nginx-load-balancer-conf
  namespace: kube-system
  resourceVersion: "1878"
  selfLink: /api/v1/namespaces/kube-system/configmaps/nginx-load-balancer-conf
  uid: d6d4c45b-188b-11e8-aae8-fa163e0ee8bb
```

```
{
  "spec": {
    "template": {
      "spec": {
        "initContainers": [
          {
            "name": "sysctl",
            "image": "alpine:3.6",
            "securityContext": {
              "privileged": true
            },
            "command": [
              "sh",
              "-c",
              "sysctl -w net.core.somaxconn=32768; sysctl -w net.ipv4.ip_local_port_range=1024 65535"
            ]
          }
        ]
      }
    }
  }
}
```

Example:
Impact of Liberty Autonomic Threading Algorithm

Java applications running on Liberty with default autonomic threading algorithm may see significant performance degradation

Note: Liberty autonomic algorithm was redesigned in 18.0.0.1 showing a 6X improvement



IBM Cloud Private Large Topology

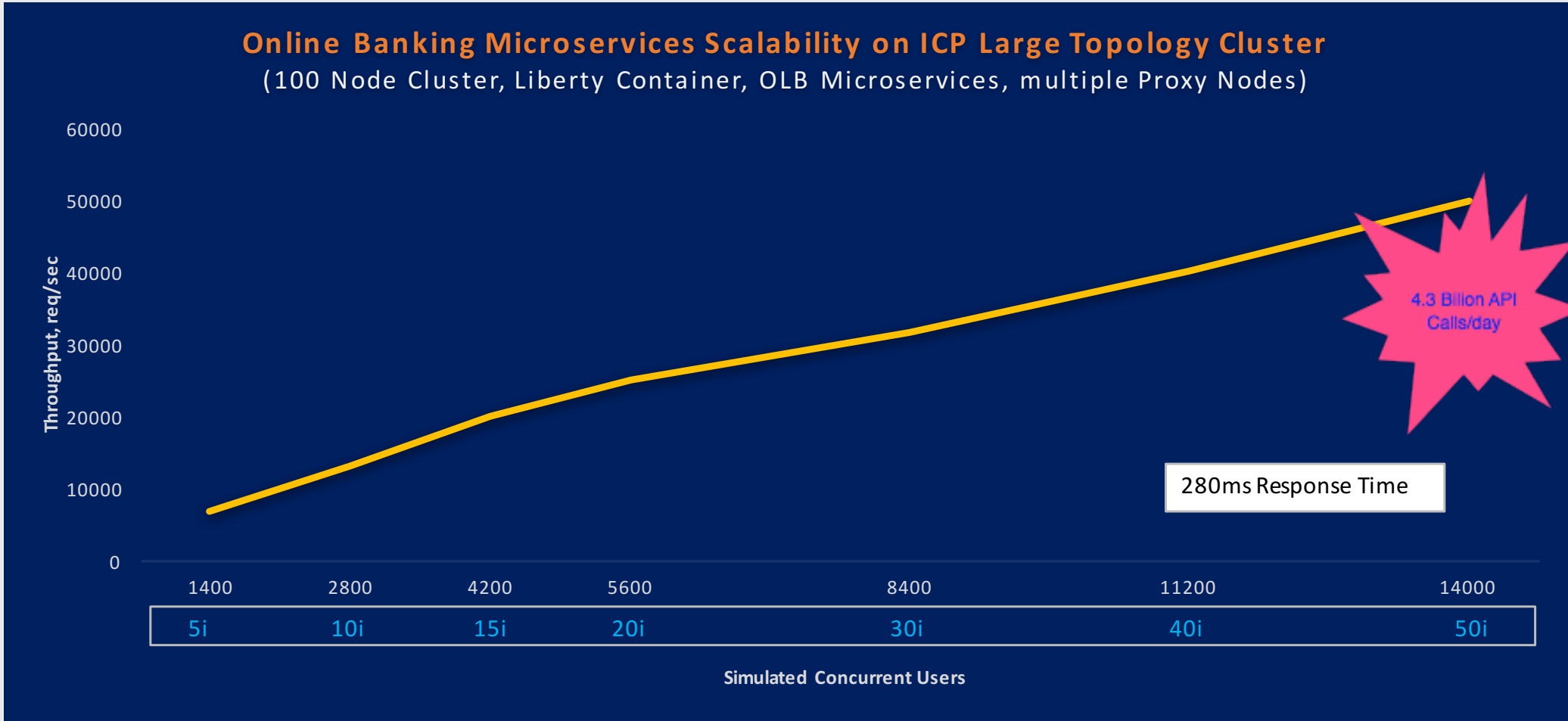
Performance in Large Topologies

Large Topology Scalability on ICP Clusters

Online Banking Application

This example is demonstrated on both 100 and 1000 node cluster running 50 instances of the application

The microservices workload scales linearly achieving 4.3 billion API calls per day



Large Topology Supported Product Limits

ICP Version	Maximum Nodes per Cluster	Total Pods Per Cluster	Pod Spec	Worker Node Spec
2.1.0.1 (Kube 1.8.3)	300	9000	0.1 CPU, 100M Memory	4 Core, 8G Memory
2.1.0.2 (Kube 1.9.1)	500	10000	0.5 CPU, 512M Memory	8 Core, 16G Memory
2.1.0.3	Target 1000	Target 20000	0.5 CPU, 512M Memory	8 Core, 16G Memory

The node sizes are representation not a recommendation with larger clusters typically accommodating larger configurations

The 2.1.0.3 values are unproven or unverified as of the time of creating this presentation

IBM v. K8s and Supported Product Limits

Kubernetes claims 5000 nodes why can ICP not?

- ICP provides extra enterprise capabilities
- ICP uses Calico as default network overlay
- ICP Calico uses node-to-node mesh to configure peering between all Calico nodes
- Calico load increases etcd load and increases greatly for very large clusters (1000+ nodes) **etcd may be broken out in its own node starting in 2.1.0.3
- Node-to-node mesh stops working if there are more than 700 nodes in a cluster

Note: It is recommended to configure route reflectors for BGP daemons in all cases starting scales approaching 1000 nodes

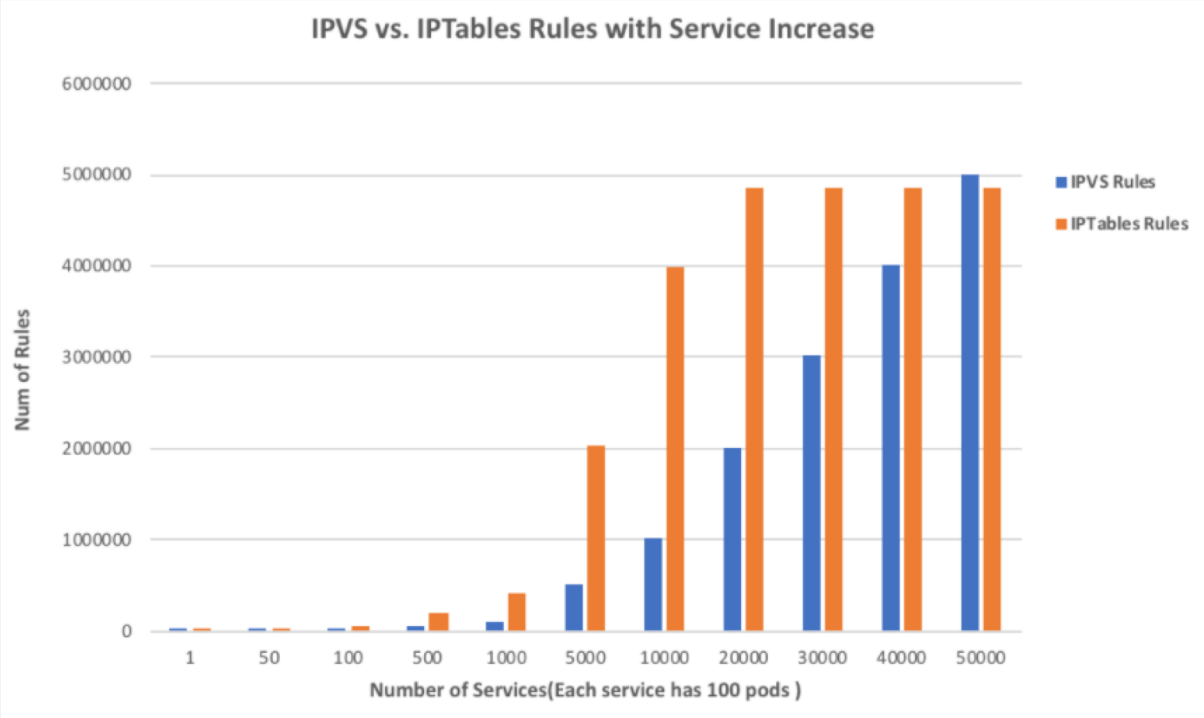
Kube-proxy: IPVS (Beta K8s 1.10, ICP 3.1.0.3)

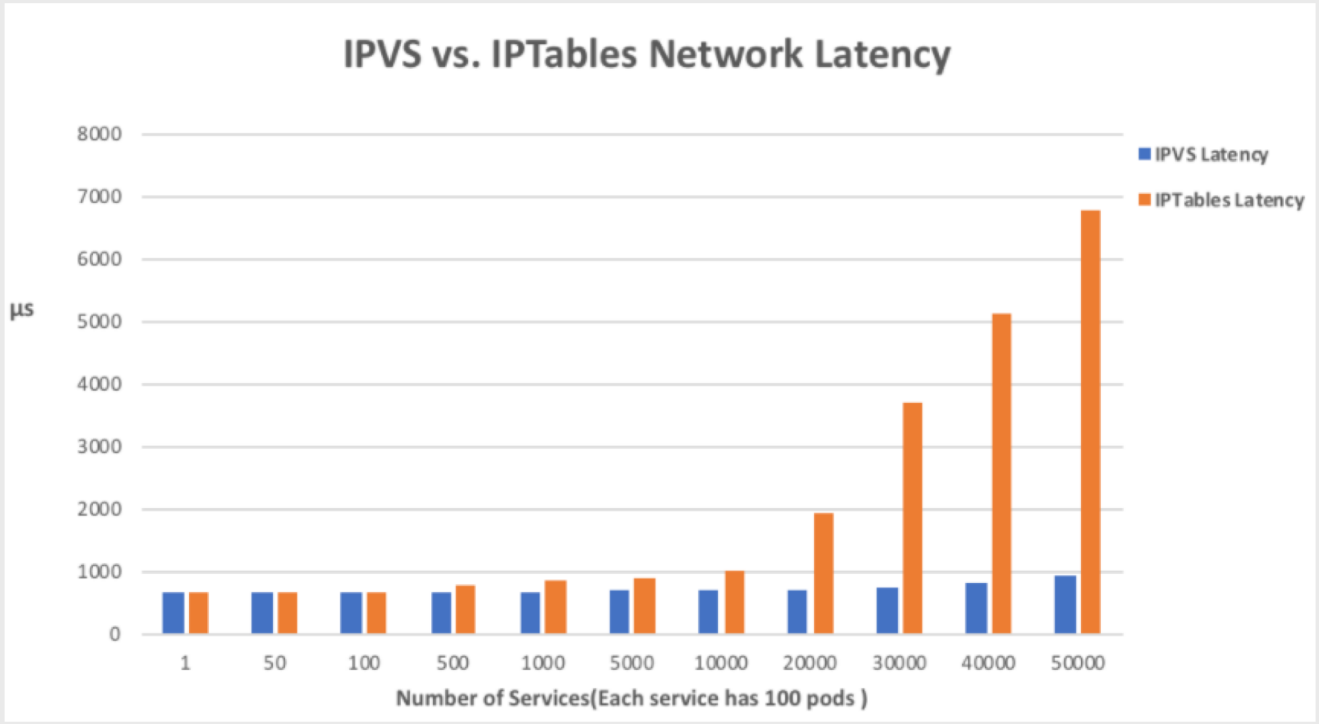
We have three choices for kube-proxy mode:
namespace, iptables (default), ipvs (beta for 2.1.0.3)

```
## Kubernetes Settings
# kube_apiserver_extra_args: []
# kube_controller_manager_extra_args: []
kube_proxy_extra_args: ["--feature-gates=SupportIPVSProxyMode=true","--proxy-mode=ipvs"]
```

Issues with iptables as a load balancers:

- Not incremental: copy all rules, make changes, save back
- Locked during update
- Time to add one rule when there are 5k services (40k rules): 11 minutes increases to 5 hours for 20k services





In Summary

Consider your workload volumes

Do you have a high number of transactions? Is the workload variable?

Consider your workload personality

Are your services network intensive? What are the programming frameworks you are using?

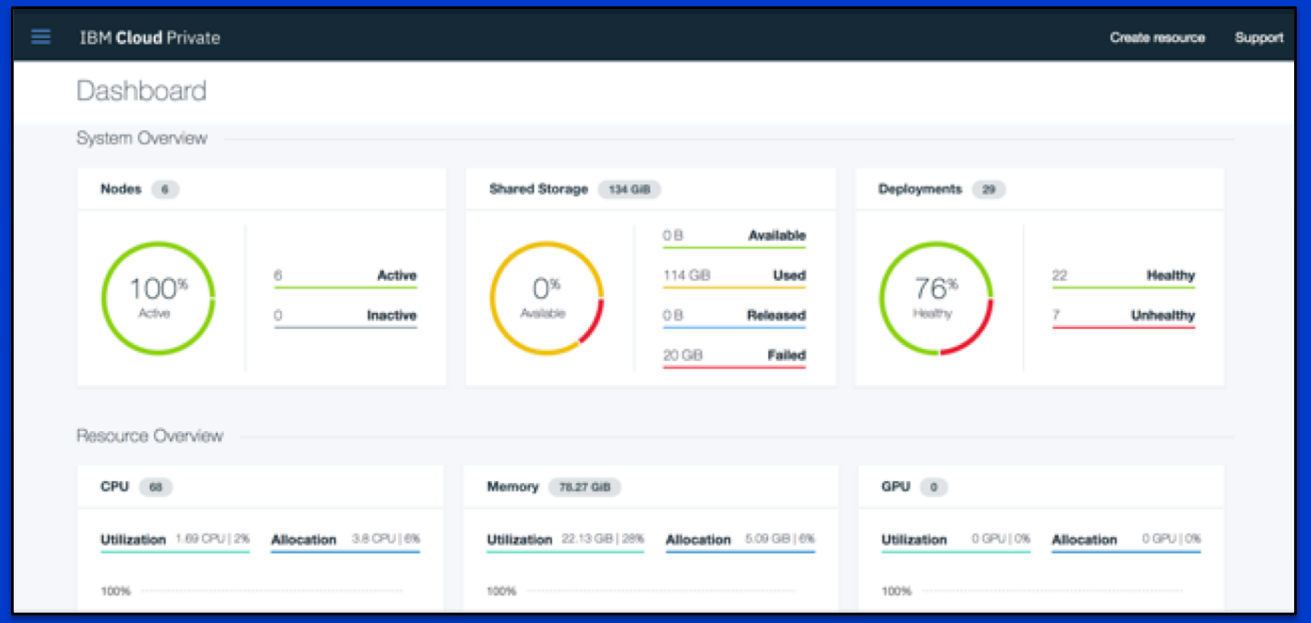
Consider your performance requirements

How do your applications scale? Horizontal? Vertical? Sharding?

Try IBM Cloud Private today!

Guided and Proof of Technology demos

Free Community Edition!



http://ibm.biz/ICP-DTE