SUSEcon digital '20

Self-driving Kubernetes Clusters

Reacting to external stimulus

TUT-1187

Flavio Castelli, Architect

Rafael Fernández López, Architect

SUSE

Agenda

1. What are we trying to solve
2. Kubernetes scaling concepts
3. Application metrics
4. Vertical Pod Autoscaler
5. Horizontal Pod Autoscaler
6. Cluster Autoscaler
7. Application design considerations

What Are We Trying To Solve

- Applications are subject to external factors: consumer load
- We don't want to have a fixed deployment:
  - Over-provisioning: expensive, it's a waste of money and resources
  - Under-provisioning: maybe fine most of the times, but can lead to the system collapsing
- Calculating the correct sizing is hard, it's better to have something dynamic

Scaling Strategies – Vertical Scaling

- Add more hardware resources, buy bigger hardware
- Doesn't require special workload architecture
- Better fit for traditional workloads

Scaling Strategies – Horizontal Scaling

- Deploy more instances of the workload
- Requires to have the proper architecture in place (more on that later)
- Better fit for cloud-native workloads

Kubernetes Pods

- Pods are the smallest deployment unit of computing for Kubernetes
- A Pod has CPU and memory requests and limits
- These limits can be specified explicitly on Pod resources, on higher level abstractions, or on the namespace the Pod runs on

Vertical Scaling Of Pods

- Change the CPU and memory requests and limits for a given Pod
- By doing this, our Pod will have more resources available
- Can be done in a live fashion against the Pod, no need to restart it

Horizontal Scaling Of Pods

- Increase the number of Pods running a certain workload
- Pods are already exposed by Kubernetes Service objects, they can act as load balancer across all the instances

Horizontal Scaling Of Pods

- Nobody schedules Pods manually
- Everybody relies on higher abstractions provided by Kubernetes:
  - A ReplicaSet manages several Pods, and among other properties, allow us to specify how many replicas of a Pod we desire to be running
  - A Deployment manages a ReplicaSet, it's a higher abstraction, it can control how replica number changes and other changes are rolled out in a running cluster

Horizontal Scaling Of Pods

- Leverage the higher-level Kubernetes objects managing Pods
- Increase or decrease the number of desired replicas of a given ReplicaSet or Deployment

We Want Automation!

- We don't want to perform the scaling manually, we want that to be automated
- How the automation should work:
  - Set a minimum number of instances that should be running all the time
  - Scale up when there's a spike of demand to avoid disruption of service
  - Scale down when the spike is over to contain the costs

Measuring External Stimulus

- We need data to automate the scale up and scale down
- Measure the impact of the external stimulus over our workload

Kubernetes Metrics Server

- Scalable solution to track all kubelets in the system
- Exposes a set of metrics in the form of time series with a well-defined API
- All metrics are kept in memory, hence, it is highly performant
- Only meant for autoscaling logic consumption
- Not meant for monitoring solutions to consume, for example

Vertical Pod Autoscaler (VPA)

- Reads information provided by the Metrics Server
- Includes a "recommender", that based on the Metrics Server data, suggests the best CPU and memory allocation
- Optionally applies the CPU and memory limits from the "recommender", vertically scaling Pods in and out

Vertical Pod Autoscaler (VPA)

```
apiVersion: autoscaling.k8s.io/v1beta2
kind: VerticalPodAutoscaler
metadata:
  name: my-application-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment
    name:       my-application
  updatePolicy:
    updateMode: "Auto"
```

Horizontal Pod Autoscaler (HPA)

- Reads information provided by the Metrics Server
- Alters the number of replicas on ReplicaSets, Deployments and StatefulSets

Horizontal Pod Autoscaler (HPA)

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: my-application-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-application
```

```
spec:
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

A Word Of Warning

- The Vertical Pod Autoscaler and the Horizontal Pod Autoscaler cannot be used together for the same set of resources
- They are incompatible with each other if they rely on the same set of metrics

How To Instruct VPA/HPA

- You can use metrics from the Metrics server (CPU and memory), that's generic
- We can do better by leveraging custom application metrics like:
  - Mean response time
  - Size of queues
  - Error rate
  - ...

Custom Metrics

- Implement a metrics-server that exposes your own defined set of metrics
- Example: leverage Prometheus through an adapter that exposes regular and custom metrics
  - Metric collector: Prometheus
  - Metric API server: Prometheus adapter
- Configure VPA or HPA to rely on these metrics

Horizontal Pod Autoscaler (HPA)

```
apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

  name: my-application-hpa

  namespace: default

spec:

  scaleTargetRef:

    apiVersion: apps/v1

    kind: Deployment

    name: my-application
```

```
spec:

  minReplicas: 1
  maxReplicas: 10

  metrics:

    - type: Pods

      pods:

        metric:

          name:
myapp_requests_per_second

        target:

          type: AverageValue

          averageValue: 2
```

What Happens When You Run Out Of Resources?

- Pods are running on computing nodes
- Computing nodes have physical constraints too
- Cluster computing resources can get exhausted too:
  - Horizontal scaling: there could be no more space to run the new Pods
  - Vertical scaling: there could be not enough space on the cluster nodes to offer the resources requested by the Pod

Cluster Autoscaler

- Set a baseline: minimum number of worker nodes part of the cluster
- Add new worker nodes to the cluster based on current demand
- Remove worker nodes once they are no longer needed

Cluster Autoscaler

- Scaling up:
  - Monitors unschedulable Pods due to insufficient resources
  - Responds by adding new worker nodes to the cluster
- Scaling down:
  - Monitors Node utilization
  - Terminates unneeded underutilized workers nodes

Cluster Autoscaler

- The current design leverages the "scaling groups" provided by all the major IaaS (AWS, Azure, GCE, OpenStack,…)
- There's work in progress to leverage the Cluster API project to have a portable and universal solution

Application Design Considerations

- Is the application micro-service oriented?
  - If not, the entire application will be scaled in and out
- Certain micro-services can increase their load depending on internal micro-service dependencies
- HPA and VPA can scale them in and out separately, in an unattended way
- Does your micro-service or application rely on specific node storage or knowledge?

Kubernetes Isn't The Silver Bullet

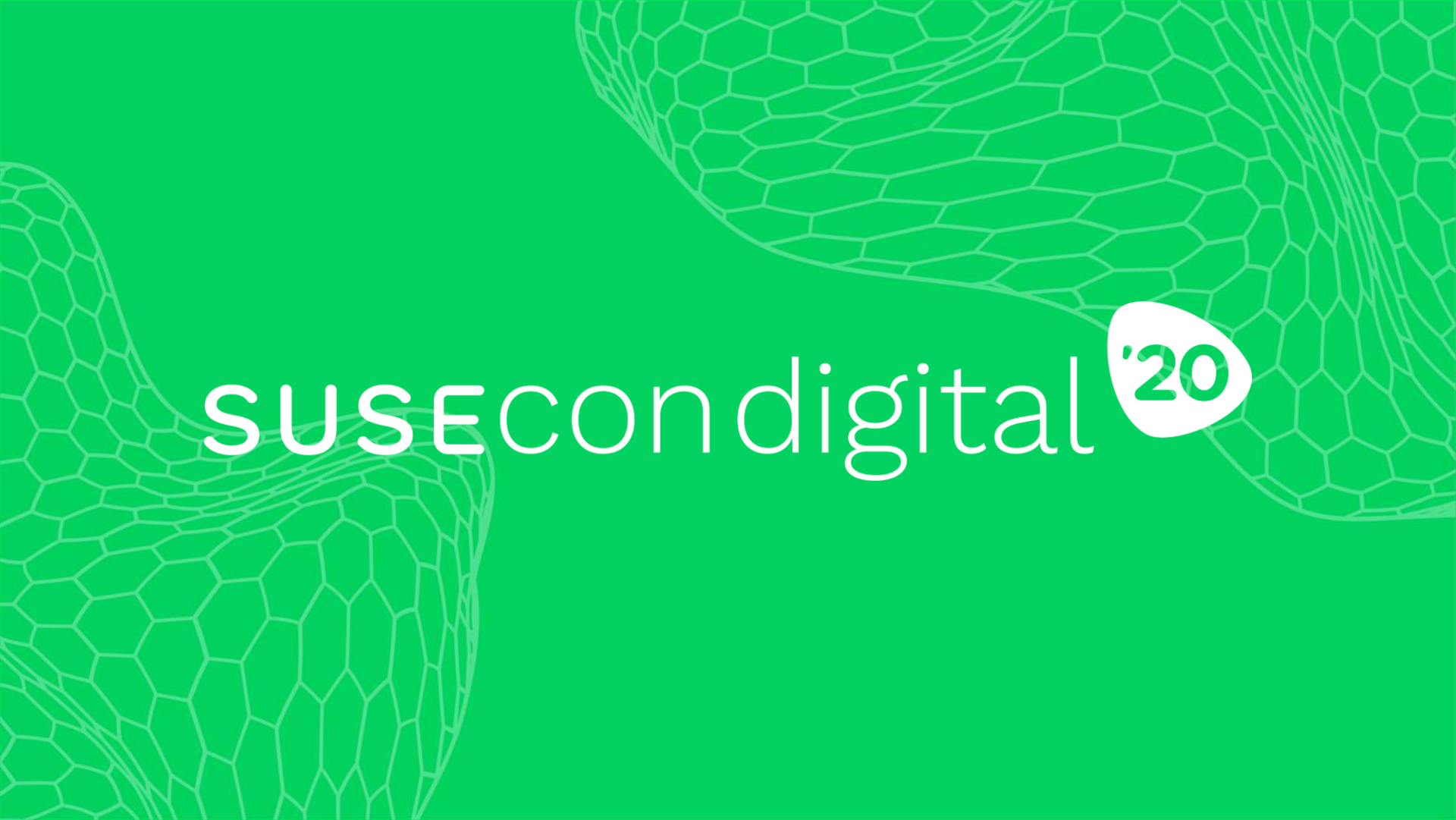
- Kubernetes provides the foundations to perform scaling in an automated way
- However your applications must be designed with this use case in mind
- Kubernetes cannot magically make your applications scale up/down

Special Applications

- Not all workloads can be scaled by simply changing the number of instances
- Some applications need certain procedures to be executed
- However, usually scaling these operations doesn't happen that often
- Examples of applications: databases

Leveraging Operators

- A Kubernetes Operator is created to handle a specific workload
- Its goal is to simplify the life cycle management of that workload
- They can simplify the installation, upgrade and scale up/down
- However they are not always consumable by HPA


SUSEcon digital '20

Q&A

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of SUSE, LLC, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.