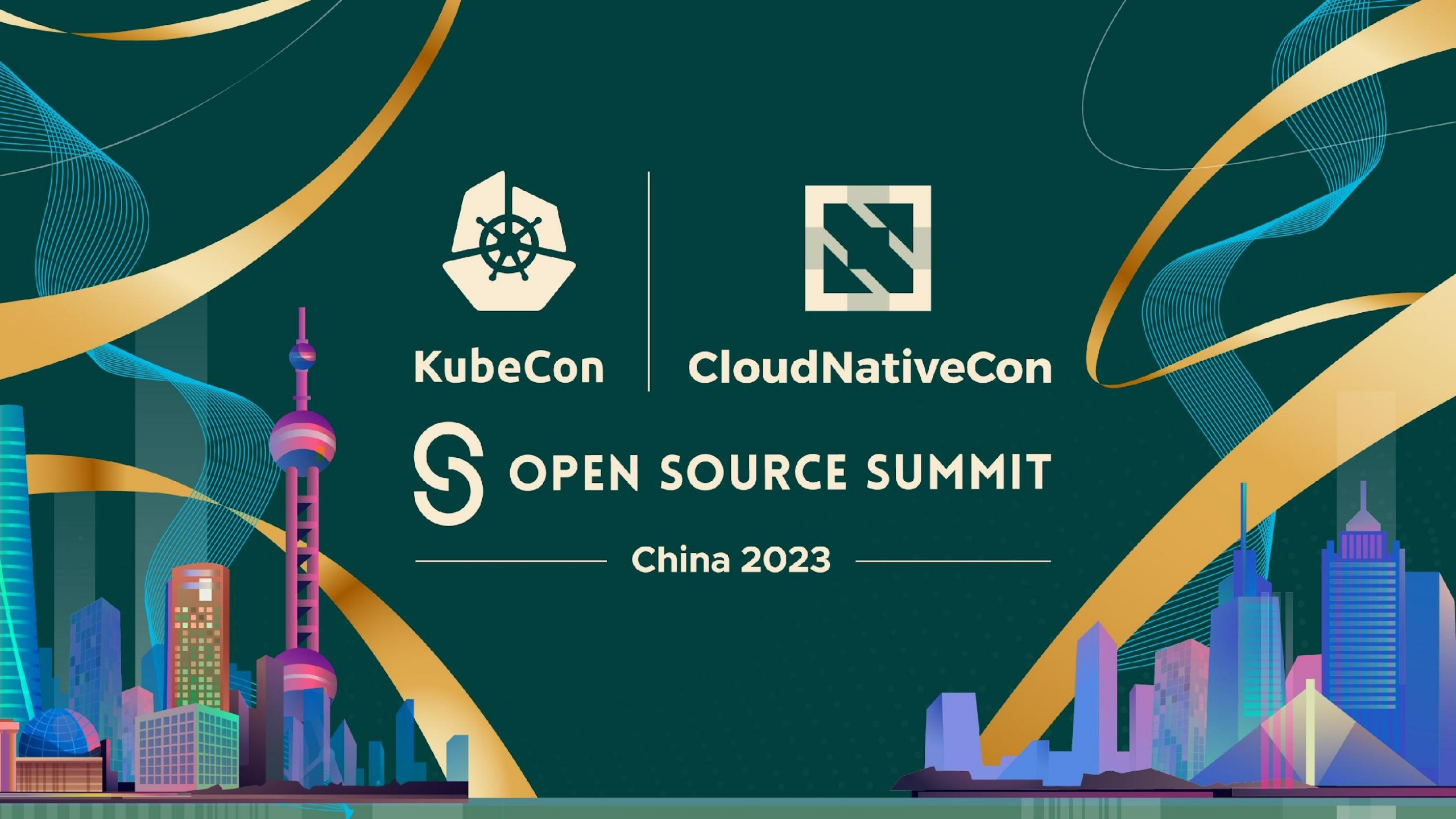

KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2023


KubeCon | CloudNativeCon

OPEN SOURCE SUMMIT

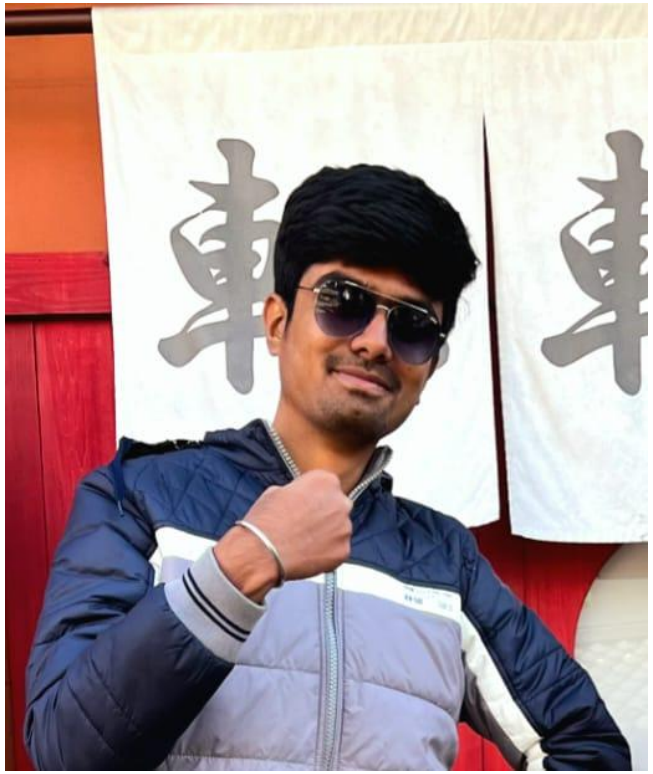
China 2023

Revolutionizing Kubernetes Logging: Structured Logs for Enhanced Monitoring

~ *Naman Lakhwani*

About Me!

- Final year University student
- Ex-Intern VMware
- Google summer of code student 2021 @CNCF (Thanos)
- Linux Foundation Mentee 2021 @Kubernetes
- GitHub: Namanl2001
- Twitter (X): lakhwani_jii



~ Naman Lakhwani

Agenda

- Logging
- Structured Logging
  - Proposal
  - Goals/Non-Goals
  - Code examples
  - why JSON structure?
  - Performance
- Contextual Logging
  - Use cases
  - Design decisions
  - Motivation
  - Goals/Non-Goals
  - Code examples

Logging

Kubernetes logging is a crucial aspect of managing containerized applications within a Kubernetes cluster.

It helps us monitor the health and performance of our applications, troubleshoot issues, and gain insights into what's happening within our cluster.

Log Sources: Kubernetes generates logs from various sources, including:

- **Application Containers**
- **Kubernetes System Components**
- **Node-Level Logs**

Logging

Log Aggregation: Aggregating logs from multiple sources and centralizing them in a single location simplifies log analysis and troubleshooting. This is often done using tools like Elasticsearch, which provides powerful searching and filtering capabilities.

Log Analysis and Visualization: Analyze logs to identify issues, troubleshoot problems, and gain insights into application behavior.

Alerting and Monitoring: Set up alerting rules to trigger notifications when specific log events or patterns are detected. Tools like Prometheus and Grafana are often used for monitoring and alerting in Kubernetes clusters.

Logging

Log Formatting: Containers typically generate logs in various formats, such as JSON, plain text, or structured data. It's essential to have a consistent log format to make log parsing and analysis easier.

To have all these features like easier Log aggregation, analysing, alerting and monitoring, we need to have structured logging format.

Structured Logging

Motivation: logging in the Kubernetes control plane didn't guarantee any uniform structure for log messages and references to Kubernetes objects in those logs.

which made parsing, processing, storing, querying and analyzing logs hard and forces administrators and developers to rely on ad-hoc solutions in most cases based on some regular expressions.

Due to those problems any analytical solution based on those logs was hard to implement and maintain.

Structured Logging

Proposal

- Define standard structure for Kubernetes log messages
- Add methods to klog to enforce this structure
- Add ability to configure Kubernetes components to produce logs in JSON format

Structured Logging

Goals:

- Make most common **logs more queryable** by standardizing log message and references to Kubernetes objects (Pods, Nodes etc.)
- Enforce log structure by introduction of new klog methods that could be used to generate structured logs.
- Simplify ingestion of logs into third party logging solutions by adding an option to output logs in the JSON format

Structured Logging

Non-Goals:

- **Not** replacing currently used logging library (klog) or the way in which it is used
- **Not** doing structuring of all logs in Kubernetes, instead focusing on improving the querability of most common logs.

Structured Logging

New Klog Methods: For each format method (Infof, Errorf) we added matching structured method (InfoS, ErrorS).

Declaration:

```
func InfoS(msg string, keysAndValues ...interface{})
```

Example:

```
klog.InfoS("Pod status updated", "pod", "kubedns", "status", "ready")
```

Result:

controller_utils.go:116] "Pod status updated" pod="kubedns" status="ready"

Structured Logging

Declaration:

```
func ErrorS(err error, msg string, keysAndValues ...interface{})
```

Example:

```
klog.ErrorS(err, "Failed to update pod status")
```

Result:

controller_utils.go:114] "Failed to update pod status" err="timeout"

Structured Logging

Log message structure

- There could be multiple ways of standardising the logging structure in Kubernetes, the one we agreed to implement in the KEP is to have following logging structure

<message> <key1>=<value1> <key2>=<value2> …

e.g.

```
pod := corev1.Pod{Name: "kubedns", Namespace: "kube-system", ...}
klog.InfoS("Pod status updated", "pod", klog.KObj(pod), "status", "ready")
```

Structured Logging

References to Kubernetes objects:

- The idea is to use k8s api first approach to get k8s objects and embed the object related information into the logs

```
func KObj (obj ObjectMeta) ObjectRef
func KRef (namespace, name string) ObjectRef

type ObjectRef struct {
  Name        string `json:"name"`
  Namespace   string `json:"namespace,omitempty"`
}
```

Structured Logging

Example:

```
pod := corev1.Pod{Name: "kubedns", Namespace: "kube-system", ...}
klog.InfoS("Pod status updated", "pod", klog.KObj(pod), "status", "ready")
```

AND

```
klog.InfoS("Pod status updated", "pod", klog.KRef("kube-system", "kubedns"),
"status", "ready")
```

Will result in below log:

controller_utils.go:116] "Pod status updated" ***pod="kube-system/kubedns"*** *status="ready"*

Structured Logging

Introduce **JSON output format** in klog:

- Introduction of new methods to klog library to support JSON.
- With klog v2 we can take further advantage of this fact and add an option to produce structured logs in JSON format.

Some **pros** of using JSON:

- Broadly adopted by logging libraries with very efficient implementations.
- Out of the box support by many logging backends
- Easily parsable and transformable
- Existing tools for ad-hoc analysis (jq)

Structured Logging


```
pod := corev1.Pod {Name: "kubedns", Namespace: "kube-system", ...}
klog.InfoS ("Pod status updated", "pod", klog.KObj(pod), "status", "ready")
```


```
{
   "ts": 1580306777.04728,
   "v": 4,
   "msg": "Pod status updated",
   "pod":{
     "name": "kubedns",
     "namespace": "kube-system"
   },
   "status": "ready"
}
```

Structured Logging

Logging performance with the new implementation. Performance is wrt to log volume and the performance impact.

logger	time [ns/op]	bytes[B/op]	allocations[alloc/op]
Text Infof	2252	248	3
Text InfoS	2455	280	3
JSON Infof	1406	19	1
JSON InfoS	319	67	1

- For text, InfoS implementation is 9% slower than Infof. Should not impact Kubernetes performance as logging takes less than 2% of overall CPU usage.
- For JSON, InfoS implementation is 77% faster than Infof

Contextual Logging

- Contextual logging is based on the go-logr API.
- The key idea is that libraries are passed a logger instance by their caller and use that for logging instead of accessing a global logger.
- The go-logr API is designed around structured logging and supports attaching additional information to a logger.

Contextual Logging

Use Cases:

- The caller can attach additional information to a logger:
  - WithName adds a prefix
  - WithValues adds key/value pairs

  When passing this extended logger into a function that uses it instead of the global logger, without having to **modify** the code that generates the log entries.

- When running unit tests, log output can be associated with the current test. Then when a test fails, only the log output of the failed test gets shown by go test.

Contextual Logging

- Attach key/value pairs that get included in all log messages
- Add names that describe which component or operation triggered a log messages
- Reduce the amount of log messages emitted by a callee by changing the verbosity

Contextual Logging

Design decisions

- One of the design decisions for contextual logging was to allow **attaching a logger** as value to a context.Context.

- When a library supports contextual logging and retrieves a logger from its context, it will still work in a binary that does not initialize contextual logging because it will get a logger that logs through klog.

Contextual Logging

Motivation - User story:

kube-scheduler developer Joan wants to know which pod and which operation and scheduler plugin log messages are associated with.

When kube-scheduler starts processing a pod, it creates a new logger with logger.WithValue("pod", klog.KObj(pod)) and passes that around.

This adds a prefix to each log message, like for example *NominatedPods/Filter/VolumeBinding*

Contextual Logging

Goals:

- Remove direct log calls through the k8s.io/klog API and the hard dependency on the klog logging implementation from all packages.
- Grant the caller of a function control over logging inside that function.

Non-Goals:

- Remove the klog text output format.
- Deprecate klog.

Contextual Logging

Risks and Mitigations:

- Uninitialized logger
- Logging during initialization
- Performance overhead

logger can be passed into the function in one of two ways:

- as explicit parameter
- attached to a context.Context

Contextual Logging

Code example:

```
-func (g *genericScheduler) snapshot() error {
+func (g *genericScheduler) snapshot(logger klog.Logger) error {
```

```
-       if err := g.snapshot(); err != nil {
+       logger := klog.FromContext(ctx)
+       logger = klog.LoggerWithValues(logger, "pod", klog.KObj(pod))
+       ctx = klog.NewContext(ctx, logger)
+
+       if err := g.snapshot(logger); err != nil {
```

Contextual Logging

Output Results:

- Here is log output from kube-scheduler -v5 for a Pod with an inline volume which cannot be created because storage is exhausted:

  Scheduler.go:436] "Attempting to schedule pod" pod="default/my-csi-app-inline-volume"
  binder.go:730] PreFilter/VolumeBinding: "PVC is not bound"
  pod="default/my-csi-app-inline-volume" pvc="default/my-csi-app-inline-volume-my-csi-volume"

- In the below log which is **not** converted to contextual logging It's not clear in which context that message gets emitted:

  csi.go:222] **"Persistent volume had no name for claim" PVC="default/my-csi-app-inline-volume-my-csi-volume"**

Current Status

Structured logging is in **GA**

Contextual logging is ready to get promoted to **Beta**

https://github.com/kubernetes/enhancements/pull/4219

How to get involved?

- Join #wg-structured-logging channel on slack, Join the mailing list
- Biweekly Meeting: Thursdays at 15:30 British time
- Meeting notes and Agenda, Meeting recordings.
- Pick up migration issues or review PRs search for label *wg-structured-logging*.

References

- KEP-1602: Structured logging
- KEP-3077: contextual logging
- Structured and Contextual Logging migration instructions
- Kubernetes contributors blog: Contextual Logging

Thank You!