LOGGING AND TROUBLESHOOTING

Logging and Troubleshooting

Basic Troubleshooting Steps

The troubleshooting flow should start with the obvious. If there are errors from the command line, investigate them first. The symptoms of the issue will probably determine the next step to check. Working from the application running inside a container to the cluster as a whole may be a good idea. The application may have a shell you can use. For example, see the following commands:

```
$ kubectl create deploy busybox --image=busybox --command sleep 3600
```

```
$ kubectl exec -ti <busybox_pod> -- /bin/sh
```

If the Pod is running, use **`kubectl logs pod-name`** to view the standard out of the container. Without logs, you may consider deploying a sidecar container in the Pod to generate and handle logging. The next place to check is networking, including DNS, firewalls and general connectivity, using standard Linux commands and tools.

Security settings can also be a challenge. RBAC, covered in the security chapter, provides mandatory or discretionary access control in a granular manner. SELinux and AppArmor are also common issues, especially with network-centric applications.

A newer feature of Kubernetes is the ability to enable auditing for the kube-apiserver, which can allow a view into actions after the API call has been accepted.

The issues found with a decoupled system like Kubernetes are similar to those of a traditional datacenter, plus the added layers of Kubernetes controllers:

- Errors from the command line
- Pod logs and state of Pods
- Use shell to troubleshoot Pod DNS and network
- Check node logs for errors, make sure there are enough resources allocated
- RBAC, SELinux or AppArmor for security settings
- API calls to and from controllers to kube-apiserver
- Enable auditing
- Inter-node network issues, DNS and firewall
- Control Plane server controllers (control Pods in pending or error state, errors in log files, sufficient resources, etc).