Managing Bro Deployments at Scale Using DevOps Technologies

Ed Sealing
Daniel Lohin
2015 Berkley Labs 100G Bro Cluster

56 Node Bro Cluster
Paper: [http://go.lbl.gov/100g](http://go.lbl.gov/100g)
“Come on, this can’t be THAT hard…”

CONCEPT:
- Build Once, deploy anywhere
- Multi-Tenancy with resource segregation
- Shared Rules across mass cluster
- Shared Resources across different tools
Our journey to enlightenment

Dec 2016- Can we put Bro in a container and get decent performance?

Summer 2017- Can we automate deployment?

Summer 2018- Can we automate a scalable deployment?
Why Containers and not VMs?

- Lightweight, stand-alone software that includes system tools, **system libraries** executable package.
- Packaged software for development, shipment as well as deployment
- **Containers share the machine’s OS kernel**
- Containers are isolated using namespaces
  - PID
  - Networking
  - Mount Points
  - UID/GID
  - Limit processors and memory
  - And more!
DevOps Principals

- Self-Service Configuration
- Automated Provisioning
- Continuous Build
- Continuous Delivery
- Continuous Integration
- Automated Release Management
- Incremental Testing

DevOps
Phase 1: Containerized Sensors perform?

• Chose two open-source network sensors (Bro & Suricata) and build DockerFiles for them
  • https://github.com/sealingtech/EDCOP-BRO
  • https://github.com/sealingtech/EDCOP-SURICATA

• What is the performance impact of running inside of a container?

• This image can be deployed again and again on different systems
• A lot of time was spent solving - How do we best get traffic to it?
# Networking options we tried

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Downside?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host Networking</strong></td>
<td>Give a container access to all networking on the physical host</td>
<td>Network isolation is gone. Container has complete control over all host networking.</td>
</tr>
<tr>
<td><strong>MacVLAN/MacVTAP</strong></td>
<td>Build to a physical interface and then connect a virtual interface to that bridge</td>
<td>Performance overhead</td>
</tr>
<tr>
<td><strong>OpenVswitch</strong></td>
<td>Build an openvswitch bridge and then create an interface with ovs-docker</td>
<td>Performance overhead and more complication</td>
</tr>
<tr>
<td><strong>SR-IOV</strong></td>
<td>Create a virtual NIC (called a Virtual Function) inside of the network card</td>
<td>Hardware dependent on this feature</td>
</tr>
</tbody>
</table>
Lessons learned

• Hardware still matters… We still need to worry about IRQs, CPU pinning, NUMA nodes and all those other complicated things

• Containers are great for when you need to build an application on a single host, but what happens when you need to scale out to multiple hosts?

• We still didn’t have integration with a larger architecture figured out (i.e. Bro feeding a Logging solution)... we needed more....

• Github or it didn’t happen! https://github.com/sealingtech/bro-docker
Multi-stage containers

Build Container

Step 1. Install all build tools (GCC, Make, bro-pkg, etc)
Step 2. Build Bro
Step 3. Build all Bro Packages
Step 4. Start up the final image

Final image

Step 1. Install packages only need to run Bro
Step 2. Copy final output of Bro from the build container
Step 3. Throw away the build container

- Bro can be built to get better performance
- Some Bro-packages require build tools
- Allows for containers to be smaller and prevents you from having to clean up!

https://github.com/dlohin/EDCOP-BRO/blob/master/container/Dockerfile
Phase 1 Progress

DevOps

- Incremental Testing
- Automated Release Management
- Continuous Delivery
- Continuous Integration
- Self-Service Configuration
- Automated Provisioning
- Continuous Build
- Continuous Integration
- Continuous Delivery
- Automated Release Management
- Incremental Testing
Phase 2: Automate an infrastructure around Bro

- Question: Now that we have a portable container, can we automatically deploy infrastructure around it?
- Answer: Yes! Our original proof-of-concept utilized Rancher to deploy Kubernetes and Bro.

<table>
<thead>
<tr>
<th>Rancher Pros and Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros:</strong></td>
</tr>
<tr>
<td>- Automatic infrastructure setup</td>
</tr>
<tr>
<td>- Simple, easy to use</td>
</tr>
<tr>
<td>- Variety of orchestrations supported</td>
</tr>
<tr>
<td>- Could connect multiple nodes now!</td>
</tr>
</tbody>
</table>
Proof of concept design
Lessons learned

• We were getting closer, but Rancher was designed to be flexible not customizable.

• The overlay network that Rancher used was a little interesting.

• Rancher was used to deploy Kubernetes, I call this rancher-caption.. It is two container management solutions on top of one another.

• NOTE: Rancher has changed a lot with 2.0, so I can’t say if it has gotten better. They have moved to a more native Kubernetes platform.

10/22/2018
Phase 2 Progress

DevOps

- Incremental Testing
- Automated Release Management
- Continuous Delivery
- Continuous Integration
- Self-Service Configuration
- Automated Provisioning
- Continuous Build
Phase 3: Build a scalable, customizable architecture

- We have containerized Bro and other sensors as well as the architecture around it

- Requirements
  - Need to be able to scale out, add more computers and applications can scale out accordingly
  - Traffic needs to be load balanced to allow sensors to scale
  - Services need to be customizable by end users

- Ability to utilize DevOps best practices
What it looks like…
Problem 1: Multi-NIC containers

- By default, Kubernetes assumes you will have one network interface per pod
- Multus (an Intel project) allows multiple ETHs per pod on different networks
Traffic Acquisition
Jenkins Auto-Build of Bro using HELM

<table>
<thead>
<tr>
<th>Clone repository</th>
<th>Build image</th>
<th>Push image</th>
<th>helm list</th>
<th>helm deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>7s</td>
<td>600ms</td>
<td>7s</td>
<td>317ms</td>
<td>1s</td>
</tr>
<tr>
<td>465ms</td>
<td>888ms</td>
<td>2s</td>
<td>315ms</td>
<td>1s</td>
</tr>
<tr>
<td>527ms</td>
<td>758ms</td>
<td>15s</td>
<td>322ms</td>
<td>2s</td>
</tr>
<tr>
<td>511ms</td>
<td>817ms</td>
<td>6s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30s (failed)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30s (failed)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>787ms</td>
<td>888ms</td>
<td>14s</td>
<td>312ms</td>
<td>1s</td>
</tr>
</tbody>
</table>
Deployment Options

### Standalone Mode

- **Pod**
  - **Elasticsearch**
  - **Statefulset**

- **Elasticsearch service**

- **Minion Host**
  - Node label: sensor=true
  - data=true
  - ingest=true
  - infrastructure=true

### Cluster Mode

- **Pod**
  - **Elasticsearch**
  - **Statefulset**

- **Data Host**
  - Node label: data=true

- **Pod**
  - **Logstash**
  - **Deployment**
  - **Ingest Host**
    - Node label: ingest=true

- **Pod**
  - **Redis service**

- **Pod**
  - **Bro**

### External Mode

- **Pod**
  - **Redis**
  - **Daemonset**

- **Pod**
  - **Redig service**

- **Pod**
  - **Bro**

- **Pod**
  - **Sensor Host**
    - Node label: sensor=true

10/22/2018
Compute resource management

[Diagram showing CPU cores with different markings indicating Interrupts and NIC, Suricata, Bro, Kernel Scheduled, and Not scheduled.]
EDCOP marketplace gives DCO a way to deploy and manage tools. EDCOP uses industry standard containerization and Infrastructure as Code (IaC) concepts to automate the deployment of software and integrate with the networking, storage and compute.

EDCOP uses automated build processes using Docker, Helm and Jenkins for managing the full lifecycle of software to allow us to build and deploy rapidly with each change of software.

Tools are integrated together through code to work together in the development process, not tacked on. Sensors immediately begin sending data to the data layer, dashboards and analytics can be applied.

Tools are delivered to users through the use of central repositories, these repositories can be shared with other members of the DCO community.

Tools are version controlled, tracked and can be updated or rolled back rapidly.

Configuration Management and Information Assurance is built into the development process.

Changes are frequent, but smaller. Dev, test and pre-prod are identical. Versions can be rapidly rolled back in situations where there are errors.
Various iterations of testing
Lessons learned

• The Kubernetes community is moving incredibly quickly, every week there is some new cool way to do things... you can get caught chasing technology
• Designing an infrastructure around Kubernetes is a change in thinking. You learn to treat applications as temporary
• Stateless apps are a lot easier to handle then stateful apps
• Bro works great inside of Kubernetes you just need to plan
Show me the Github!!

- Website: https://edcop.io
- EDCOP Deployment Platform: https://github.com/sealingtech/EDCOP
- BRO: https://github.com/sealingtech/EDCOP-BRO
- All the other components are in separate repos, just look for EDCOP-<tool name> here: https://github.com/sealingtech/
- Contact us:
  - ed.sealing@sealingtech.com
  - daniel.lohin@sealingtech.com