Building Operable Systems

Mark Imbriaco
VP, Technical Operations
DigitalOcean

Operability?

Operability is the ability to keep … a system … in a safe and reliable functioning condition, according to pre-defined operational requirements.

In a computing systems environment with multiple systems this includes the ability of products, systems and business processes to work together to accomplish a common task such as finding and returning availability of inventory for flight.

Collaboration and Visibility

Runtime Consistency

- The environment where the software operates must be consistent.

- This consistency is important both from one deploy to the next as well as across all nodes in a larger system.

Configuration Management

- Tools like Puppet, Chef, cfengine, Ansible, etc.
- With great power comes great responsibility.
  - Decoupling can be a problem.
  - Deploying via config management is not ideal of cycle time or predictable timing.
  - Avoid managing application dependencies with configuration management as much as possible.

Golden Images

- Strong consistency.
- Immutable infrastructure is appealing.
- Less suitable for applications that have to maintain local state.
- Containers are an interesting hybrid.

Metric Collection

- “If it moves, graph it.” - Etsy
- Business metrics alongside system metrics.
- Frequency: As often as possible.
- Resolution: As high as possible.
- Retention: As long as possible.

Metric Reporting

- Avoid information overload by keeping dashboards simple.
- Support intuition and pattern recognition for key metrics.
- Make collaboration simple. Chat integration, permalinks.
- Alert from metric values.

Logging

- More is better, sometimes.
- Default to informational, with easy mechanism for changing log level in running system.
- Standardize log generation and collection across system components.
- Logs are a data stream, and format plays a huge role in how well you can gain value from it.
- Use unique identifiers to allow tracing.

Log Formats

- Structure for both human scanning and machine parsing. Hint: JSON is not scannable.

- Simple key/value pairs in a predictable order is a great technique.

```
Aug 12 00:02:11 node123 velocityd[1234]: id=1234 action=execute
   task=add_review args=imbriaco,3 at=start

Aug 12 00:02:14 node123 velocityd[1234]: id=1234 action=execute
   task=add_review args=imbriaco,3 at=end
```

Process Inspection

- Frequently seen as both the first and last resort.
- Lots of tools available from simple like ps to complex like sysdig, with wide variety in between.
- Most tools are focus on the operating system view rather than the application context.
- Both passive inspection and active inspection are important.

Passive Process Inspection

- There is a lot of very valuable information in the process list, but it is an under-utilized resource.

```
% ps auxw|head -1
USER               PID  %CPU %MEM      VSZ    RSS  TTY   STAT STARTED      TIME COMMAND
 % ps auxw|grep velocityd
mark              1367   0.0  0.2  2465840   9880 ?     S+    6:02PM   0:00.10 velocityd
mark              1736   0.0  0.0  2432784    608 s002  S+   10:07PM   0:00.00 grep
velocityd
```

- Use the process name field for fun and profit.

```
velocityd: id=1234 action=execute task=add_review args=imbriaco,3 at=start
```

Active Process Inspection

- Allow introspection into the current running state of the process.

- Simple HTTP based health check endpoints provide a lot of leverage.

```
{
  "status": "OK",
  "active_workers": 10,
  "available_workers": 5,
  "queue_depth": 0
}
```

- Signal handling for dumping state.

Resilience Patterns

- Feature Flags & Graceful Degradation
- Circuit Breakers
- Ubiquitous Timeouts
- Backpressure

Feature Flags

- Decorate code with conditionals that allow you to turn features on and off.
- Not just for development!
- Define critical paths and use feature flags to protect them.

Circuit Breakers

- Like feature flags, but automated.
- Can have a variety of triggering conditions such as library call volume, response time, overall system load.
- Metrics are key.

Timeouts

- Critical for addressing outliers in hot code paths.
- Every external resource call should have a timeout.
- Different timeout categories: Consider connection, request, response, etc.

Backpressure

- Timeouts and errors can quickly cause thundering herds.

- Signaling mechanism to tell client when and how it can retry.

- Client-side back-off with server-side hinting.

Operability Reviews

- Written guidelines and operational standards go a long way.
- Embedding operations in engineering projects for ongoing review.
- Go/No-Go Meetings - Everybody gets a vote, anyone can stop the launch.
- Engineering writes the initial run books, supports release in production initially.

Failure Testing

- Runbooks should come with failure simulation steps for validation.
- Runbook validation should be regularly scheduled.
- Executing failure simulation and remediation in a test environment is a great training tool.
- Automated fault injection with tools like Chaos Monkey is a fantastic forcing function.
- Testing larger scale failures in Gameday exercises improves confidence and uncovers latent faults.

Post-Mortems

- Blame doesn't help.
- Look at systemic issues, not for root cause.
- Actionable - Put due dates on remediation items and track them.
- Focus on improvement.

Selected Resources

- Release It! Design and Deploy Production Ready Software
  Michael Nygard

- The Field Guide to Understanding Human Error
  Sydney Dekker

- Resilience Engineering In Practice: A Guidebook
  Erik Hollnagel, et al.

- Web Operations
  John Allspaw, Jesse Robbins (Editors)