Layers
Yelp’s Mission
Connecting people with great local businesses.
Going The Extra Mile: This Heroic Ambulance Driver Drove For An Extra 2 Hours To Find A Hospital With 4.5 Stars On Yelp
(Abstraction) Layers
(related: separation of concerns)
Abstractions?

"Being abstract is something profoundly different from being vague ... The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise." - Edsger Dijkstra

All non-trivial abstractions, to some degree, are leaky. - Joel Spolsky

Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem - David Wheeler
Danger: I don’t know as much as Rich Hickey

Summary
Rich Hickey emphasizes simplicity's virtues over easiness, showing that while many choose easiness they may end up with complexity, and the better way is to choose easiness along the simplicity path.

Bio
Rich Hickey, the author of Clojure, is an independent software designer, consultant and application architect with over 20 years of experience in all facets of software development. Rich has worked on scheduling systems, broadcast automation, audio analysis and fingerprinting, database design, yield management, exit poll systems, and machine learning.

Compactly
- To interleave, entwine, braid
  - archaic
- Don’t do it!
- Compling things is the source of complexity
- Best to avoid in the first place
Identify the Author’s Purpose

A. **Persuade** the audience that his/her design recommendations are better than the current status quo

B. **Inform** others about the history of architecture decisions in computer science

C. **Inspire** engineers to think outside the box when designing systems

D. **Entertain** the audience with historic anecdotes

E. **None** of the above
DevOps
Traditional thinking

Dev’s job is to add new features
Ops’ job is to keep the site stable and fast

10+ Deploys Per Day: Dev and Ops Cooperation at Flickr

Published on Jun 23, 2009
Let’s Talk About Devops

- PM: Features
- UX: Wireframes
- Lead Dev: Tasks
- Devs: Code / Unit Tests
- QA: QA | Acceptance Tests
- Release Engineering: Build + Release
- Operations: Deployment | Monitoring | Alerting
“What if we got rid of some of the layers?”
IT WAS DEV!

IT WAS QA!

IT WAS OPS!
Let’s Talk About Devops (no-ops?)

- **Features**
- **Wireframes**
- **Code / Unit Tests**
- **QA | Acceptance Tests**
- **Build + Release**
- **Deployment | Monitoring | Alerting**

**PM**

**UX**

**Devs**

**Separation Of Concerns**
Why?

- Increased “Performance” (deploys per day)
- Better ownership of issues
- Harder to hire
- Massive increase in developer complexity
- Large inter-team communication overhead
Filesystems
Let’s talk about filesystems

- `/mnt/stuff/foo` files
- `/mnt/stuff` ext4
- `/dev/mapper/luks-17c8...` dm-crypt
- `/dev/mapper/lv0` LVM
- `/dev/md0` Software Raid
- `/dev/sd{a,b}` Raw Block Devices
“What if we got rid of some of the layers?”
Let’s talk about zfs
Now What?

- Better error detection/recovery
- We can deduplicate things
- More efficient snapshots / Repair
- All new tooling is not interoperable
New Concept: “Sympathetic Abstraction”

- “Opposite” of “Leaky”?
- https://mechanical-sympathy.blogspot.com/
~ $ ssh -vv leb1.xkyle.com -- true
OpenSSH_7.7p1 Ubuntu-4ubuntu0.1, OpenSSL 1.0.2n 7 Dec 2017
debg1: Reading configuration data /home/kyle/.ssh/config
debg1: /home/kyle/.ssh/config line 1: Applying options for leb1.xkyle.com
debg1: Reading configuration data /etc/ssh/ssh_config
debg1: /etc/ssh/ssh_config line 19: Applying options for *
debg2: resolving "leb1.xkyle.com" port 4242
debg2: ssh_connect_direct: needpriv 0
debg1: Connecting to leb1.xkyle.com [2607:8b00:0:96::d0eb:ec0a] port 4242.
...
debg1: Entering interactive session.
debg1: pledge: network
debg2: channel_input_open_confirmation: channel 0: callback start
debg2: fd 3 setting TCP_NODELAY
debug2: client_session2_setup: id 0
debg1: Sending environment.
debg1: Sending env LANG = en_US.UTF-8
debg2: channel 0: request env confirm 0
debg1: Sending command: true

Transferred: sent 2696, received 2516 bytes, in 0.1 seconds
Bytes per second: sent 23959.5, received 22359.8
debg1: Exit status 0
commit_failure_policy

(Default: stop) Policy for commit disk failures:

- **die**
  - Shut down gossip and Thrift and kill the JVM, so the node can be replaced.
- **stop**
  - Shut down gossip and Thrift, leaving the node effectively dead, available for inspection using JMX.
- **stop_commit**
  - Shut down the commit log, letting writes collect but continuing to service reads (as in pre-2.0.5 Cassandra).
- **ignore**
  - Ignore fatal errors and let the batches fail.

**disk_optimization_strategy**

(Default: ssd) The strategy for optimizing disk reads. Possible values: ssd or spinning.
Kyle’s Abstraction Law:

“Over time, layers tend toward becoming more sympathetic (or merged) in the pursuit of performance.”
HTTP(S)
OSI Layer Model History

- Published in 1984 under ISO7498 (35 years ago)
- Mostly by Charles Bachman from his experience with ARPANET, etc
- Written with the purpose of giving protocol designers a Map to follow

ISO/IEC 7498-1:1994
Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model

This standard was last reviewed and confirmed in 2000. Therefore this version remains current.

The electronic version of this International Standard can be downloaded from the ISO/IEC Information Technology Task Force (ITTF) website.
Application 7
Presentation 6
Session 5
Transport 4
Network 3
Datalink 2
Physical 1

Routers
Switches
Hubs

© 2000 How Stuff Works
HTTPS in the OSI Network Model

HTTP (Hypertext Transfer Protocol) → Layer 7: Application

SSL → Layer 6: Presentation

N/A (Right? HTTP is “stateless”) → Layer 5: Session

TCP (Transmission Control Protocol) → Layer 4: Transport

IP (Internet Protocol) → Layer 3: Network

802.11 (Wi-Fi?) → Layer 2: Data link

802.11 (Wi-Fi?) → Layer 1: Physical
The IETF protocol development effort is not concerned with strict layering. Some of its protocols may not fit cleanly into the OSI model, although RFCs sometimes refer to it and often use the old OSI layer numbers.

The IETF has repeatedly stated that Internet protocol and architecture development is not intended to be OSI-compliant. RFC 3439, addressing Internet architecture, contains a section entitled: "Layering Considered Harmful".
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Four layers</strong></td>
<td>Four layers</td>
<td>Five layers</td>
<td>Four+one layers</td>
<td>Five layers</td>
<td>Three layers</td>
<td>Seven layers</td>
</tr>
<tr>
<td>&quot;Internet model&quot;</td>
<td>&quot;Internet model&quot;</td>
<td>&quot;Five-layer Internet model&quot; or &quot;TCP/IP protocol suite&quot;</td>
<td>&quot;TCP/IP 5-layer reference model&quot;</td>
<td>&quot;TCP/IP model&quot;</td>
<td>&quot;Arpanet reference model&quot;</td>
<td>OSI model</td>
</tr>
<tr>
<td>Application</td>
<td>Application</td>
<td>Application</td>
<td>Application</td>
<td>Application</td>
<td>Application/Process</td>
<td>Application</td>
</tr>
<tr>
<td>Transport</td>
<td>Transport</td>
<td>Transport</td>
<td>Transport</td>
<td>Host-to-host or transport</td>
<td>Host-to-host</td>
<td>Transport</td>
</tr>
<tr>
<td>Internet</td>
<td>Internetwork</td>
<td>Network</td>
<td>Internet</td>
<td>Internet</td>
<td></td>
<td>Network</td>
</tr>
<tr>
<td>Link</td>
<td>Network interface</td>
<td>Data link</td>
<td>Data link (Network interface)</td>
<td>Network access</td>
<td>Network interface</td>
<td>Data link</td>
</tr>
<tr>
<td></td>
<td>Physical (Hardware)</td>
<td>Physical</td>
<td>Physical</td>
<td></td>
<td></td>
<td>Physical</td>
</tr>
</tbody>
</table>
HTTP in the OSI Network Model

Layer 7: Application
- HTTP (Hypertext Transfer Protocol)

Layer 6: Presentation
- SSL
- N/A (Right? HTTP is “stateless”)

Layer 5: Session
- TCP (Transmission Control Protocol)
- IP (Internet Protocol)

Layer 4: Transport
- 802.11 (Wi-Fi?)

Layer 3: Network

Layer 2: Data link

Layer 1: Physical
“What if we got rid of some of the layers?”
QUIC in the IP Model

QUIC (Quick UDP Internet Connections)

UDP (User Datagram Protocol)

IP (Internet Protocol)

802.11 (Wi-Fi)

Transport Layer (OSI 4)

Internet Layer (OSI 3)

Link Layer (OSI 1,2)

Application Layer (OSI 5,6,7)

Encryption

Sessions

Multiplexing

Error Correction

Separation of Concerns
TCP

TCP + TLS

QUIC (equivalent to TCP + TLS)

100 ms

200 ms\(^1\)
300 ms\(^2\)

0 ms\(^1\)
100 ms\(^2\)

---

1. Repeat connection
2. Never talked to server before

Cloud CDN throughput (50th percentile)
Now What?

- Roaming sessions
- 0-RRT Handshakes
- Better throughput
- Better loss recovery
- You can’t debug via tcpdump anymore!
- Only a few companies has dared to ship this!
- Very poor tooling / ecosystems around it!
- Must have faith in QUIC-crypto!
(Micro) Services
“What if we ADDED some more layers?”
Лепра
@leprasatorium
Хуепра. Добро пожаловать отсюда
Default City

Лепра @leprasatorium · 2h
Викторианские советы
Часть 2 pic.twitter.com/21PraRYBaO

Лепра @leprasatorium · 2h
Викторианские советы
Now What?

- No more atomic cross-cutting changes
- Simplified deploy process? (Simple for who?)
- Polyglot stack (easy to hire for)
- No more unified tooling
- You scale out components (scale to millions of rps)
- You scale out development (1000 engineers)
k8s
Traditional Cloud Deployment

Layer 0: IaaS Provider Bare Metal

Layer 1: IaaS VM
- c5.xlarge
- N/A

Layer 2: OS
- Ubuntu x86_64

Layer 3: Application Code
- Java Jar
“What if we ADDED EVEN MORE layers?”
Layers of a k8s Infra

Layer 0: IaaS Provider Bare Metal

Layer 1: IaaS VM
- c5.xlarge
- Ubuntu x86_64

Layer 2: OS
- pod-foo
- kublet

Layer 3: Container Orchestration
- Layer 4: Pod
- pod-foo
- kublet

Layer 5: Container
- docker

Layer 6: Process
- java -jar
K8s Layers

Layer 5: k8s federation
Prod k8s federation

Layer 4: k8s cluster
uswest1-prod

Layer 3: Deployment / ReplicaSet
Frontend replicaset

Layer 2: Pod
myapp-pod

Layer 1: Docker Container
FROM apache2

Layer 0: Process
httpd
“What if we got rid of some of the layers?”
# AWS Metal Instances

<table>
<thead>
<tr>
<th>API Name</th>
<th>Memory</th>
<th>vCPUs</th>
<th>Network Performance</th>
<th>Linux On Demand cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>Search</td>
<td>Search</td>
<td>Search</td>
<td>Search</td>
</tr>
<tr>
<td>u-9tb1.metal</td>
<td>9216.0 GiB</td>
<td>448 vCPUs</td>
<td>25 Gigabit</td>
<td>unavailable</td>
</tr>
<tr>
<td>r5d.metal</td>
<td>768.0 GiB</td>
<td>96 vCPUs</td>
<td>25 Gigabit</td>
<td>$7.776000 hourly</td>
</tr>
<tr>
<td>u-12tb1.metal</td>
<td>12288.0 GiB</td>
<td>448 vCPUs</td>
<td>25 Gigabit</td>
<td>unavailable</td>
</tr>
<tr>
<td>i3.metal</td>
<td>512.0 GiB</td>
<td>72 vCPUs</td>
<td>25 Gigabit</td>
<td>$5.504000 hourly</td>
</tr>
<tr>
<td>u-6tb1.metal</td>
<td>6144.0 GiB</td>
<td>448 vCPUs</td>
<td>25 Gigabit</td>
<td>unavailable</td>
</tr>
<tr>
<td>m5.metal</td>
<td>384.0 GiB</td>
<td>96 vCPUs</td>
<td>25 Gigabit</td>
<td>$5.376000 hourly</td>
</tr>
<tr>
<td>r5.metal</td>
<td>768.0 GiB</td>
<td>96 vCPUs</td>
<td>25 Gigabit</td>
<td>$6.720000 hourly</td>
</tr>
<tr>
<td>m5d.metal</td>
<td>384.0 GiB</td>
<td>96 vCPUs</td>
<td>25 Gigabit</td>
<td>unavailable</td>
</tr>
<tr>
<td>z1d.metal</td>
<td>384.0 GiB</td>
<td>48 vCPUs</td>
<td>25 Gigabit</td>
<td>$5.064000 hourly</td>
</tr>
</tbody>
</table>
Future Layers

Hardware → Layer 0/1: IaaS Provider Bare Metal
Future Layers

Extreme Example - Unikernels?

Layer 0/1: IaaS Provider Bare Metal

Hardware

Layer 2-7: Unikernel (No OS, just a directly booting application)

MirageOS, etc
Future Layers - AWS Lambda / FaaS?

Layer 0/1: IaaS Provider Bare Metal

Layer 2-5: Amazon / k8s Stuff

Layer 6: Container

Layer 7: Your Process

Your Code

Firecracker / Docker

Same Stuff As Before

Hardware

Layer 0/1: IaaS Provider Bare Metal
What? We Are Back To Where We Started?

- Where is the mechanical sympathy?
- What happened to Layer-Smashing, Kyle?
- **Three** Reasons why we are moving this direction...
Reason 1:
You have a layer for just the stuff you care about
Reason 2: Sympathetic routing and dispatch

Or

“Functions as a Service (Serverless)” Layer Smashing
Reason 3:
Sympathetic Compute Scheduling Layers (k8s operators)

Autoscaling Group

MySQL

? 

K8s MySQL Operator

MySQL

I know what I must do
<table>
<thead>
<tr>
<th>Operator</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPA Operator</td>
<td>Beta</td>
<td>Horizontal Pod Autoscaler operator for Kubernetes. Annotate, and let HPA operator do the rest.</td>
</tr>
<tr>
<td>Cluster Autoscaler</td>
<td>Beta</td>
<td>Manage Kubernetes cluster-autoscaler deployments</td>
</tr>
<tr>
<td>Ark</td>
<td>Beta</td>
<td>Ark is a utility for managing disaster recovery, this operator manages the backup and restoration of cluster cor</td>
</tr>
<tr>
<td>Kanister</td>
<td>Beta</td>
<td>Kanister is an extensible framework for application-level data management on Kubernetes</td>
</tr>
<tr>
<td>Airflow</td>
<td>Alpha</td>
<td>A Kubernetes operator to manage Apache Airflow.</td>
</tr>
<tr>
<td>MXNet</td>
<td>Alpha</td>
<td>Apache MXNet is a modern open-source deep learning framework used to train, and deploy deep neural networ</td>
</tr>
<tr>
<td>Spark (GCP)</td>
<td>Alpha</td>
<td>Kubernetes CRD operator for specifying and running Apache Spark applications idiomatically on Kubernetes.</td>
</tr>
<tr>
<td>Spark (radanalytics.io)</td>
<td>Beta</td>
<td>ConfigMap-based operator for deploying ephemeral Apache Spark clusters and intelligent applications that spe</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>Beta</td>
<td>Tools for ML/Tensorflow on Kubernetes.</td>
</tr>
<tr>
<td>PyTorch</td>
<td>Beta</td>
<td>PyTorch on Kubernetes</td>
</tr>
<tr>
<td>Cassandra (instaclustr)</td>
<td>Alpha</td>
<td>Cassandra is a free and open-source distributed wide column store NoSQL database management system des</td>
</tr>
<tr>
<td>Cassandra (vogkovski)</td>
<td>Alpha</td>
<td>Cassandra is a free and open-source distributed wide column store NoSQL database management system des</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>Alpha</td>
<td>Amazon DynamoDB is a proprietary NoSQL database service that supports key-value and document data struc</td>
</tr>
<tr>
<td>Keel</td>
<td>Beta</td>
<td>Kubernetes Operator to automate Helm, DaemonSet, StatefulSet &amp; Deployment updates</td>
</tr>
<tr>
<td>Flux</td>
<td>Beta</td>
<td>The GitOps Kubernetes operator</td>
</tr>
<tr>
<td>Environment Operator</td>
<td>Beta</td>
<td>Kubernetes Environment Management</td>
</tr>
<tr>
<td>Chart Operator</td>
<td>Alpha</td>
<td>The chart-operator deploys Helm charts by reconciling against a CNR registry.</td>
</tr>
<tr>
<td>CouchDB</td>
<td>Alpha</td>
<td>Prototype Kubernetes operator for couchDB</td>
</tr>
<tr>
<td>MongoDB (kbst)</td>
<td>Alpha</td>
<td>MongoDB Operator for Kubernetes</td>
</tr>
<tr>
<td>MongoDB (ultimaker)</td>
<td>Beta</td>
<td>MongoDB Operator for MongoDB Replica Sets and Backups</td>
</tr>
<tr>
<td>MongoDB (Official)</td>
<td>Beta</td>
<td>MongoDB Enterprise Operator for Kubernetes</td>
</tr>
<tr>
<td>MongoDB (Percona)</td>
<td>Beta</td>
<td>A Kubernetes operator for Percona Server for MongoDB</td>
</tr>
<tr>
<td>RethinkDB</td>
<td>Beta</td>
<td>RethinkDB is a free and open-source, distributed document-oriented database. This is a Kubernetes operator to</td>
</tr>
<tr>
<td>Couchbase (official)</td>
<td>Production</td>
<td>This is a paid product from Couchbase</td>
</tr>
<tr>
<td>ArangoDB</td>
<td>Beta</td>
<td>ArangoDB Kubernetes Operator. Start ArangoDB on Kubernetes in 5min.</td>
</tr>
<tr>
<td>svcat</td>
<td>Beta</td>
<td>Service Catalog is a Kubernetes extension API that enables applications running on Kubernetes clusters to con</td>
</tr>
<tr>
<td>AWS (Giant Swarm)</td>
<td>Production</td>
<td>Manages Giantnetes Kubernetes clusters running on Amazon Web Services</td>
</tr>
<tr>
<td>CloudFormation</td>
<td>Alpha</td>
<td>AWS CloudFormation is a service that helps you model and set up your Amazon Web Services resources. Using</td>
</tr>
<tr>
<td>KubeVirt</td>
<td>Beta</td>
<td>Kubernetes Virtualization Operator with API and runtime in order to define and manage virtual machines</td>
</tr>
<tr>
<td>OpenStack</td>
<td>Beta</td>
<td>SAP OpenStack operator creates various resources in OpenStack</td>
</tr>
<tr>
<td>VPC Peering</td>
<td>Beta</td>
<td>A Kubernetes Operator to manage the lifecycle of AWS VPC Peering Connections</td>
</tr>
<tr>
<td>AWS Service Operator (off)</td>
<td>Alpha</td>
<td>AWS Service Operator allows you to create AWS resources using kubectl</td>
</tr>
<tr>
<td>Azure Operator (Giant Sw</td>
<td>Production</td>
<td>Azure operator manages Kubernetes clusters running in Giantnetes on Azure</td>
</tr>
<tr>
<td>GCP Operator (paulczar)</td>
<td>Alpha</td>
<td>GCP operator for Kubernetes</td>
</tr>
</tbody>
</table>
Conclusions?

1. “Sympathetic Layers” are the best layers!

2. k8s operators and FaaS as re-slicing where the layers are (for the greater good!)

3. You should watch “Simple Made Easy” before changing your mind

https://www.infoq.com/presentations/Simple-Made-Easy
Hold for (shameless) applause,

Then

Questions?

SEPARATION OF CONCERNS IS OVERRATED

CHANGE MY MIND
Attribution

X Server

- X Client (GUI App)
- Window Manager
- 3d Game

Ioctl via libdrm
/dev/dri/card0

X11 Protocol over Socket
/tmp/.X11-unix/X0

Linux Kernel

GPU
“Mechanism, not policy”
“What if we got rid of some of the layers?”
Now What?

- Lower latency GUI Interactions
- NO TEARING