Quilt

Ethan J. Jackson, Aurojit Panda, Kevin Lin,
Johann Schleier-Smith, Nicholas Sun, Luise Valentin,
Yuen Mei Wan, Scott Shenker

quilt.io

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

Everything has an API

Compute

amazon web services™ EC2

Google Cloud Platform

openstack™

Microsoft Azure

DigitalOcean

docker SWARM

MESOS

Nomad

kubernetes by Google

Network

socketplane

weaveworks

vmware®
NSX

OVN
(Open Virtual Network)

OPEN
DAYLIGHT

PROJECT
CALICO

DevOps

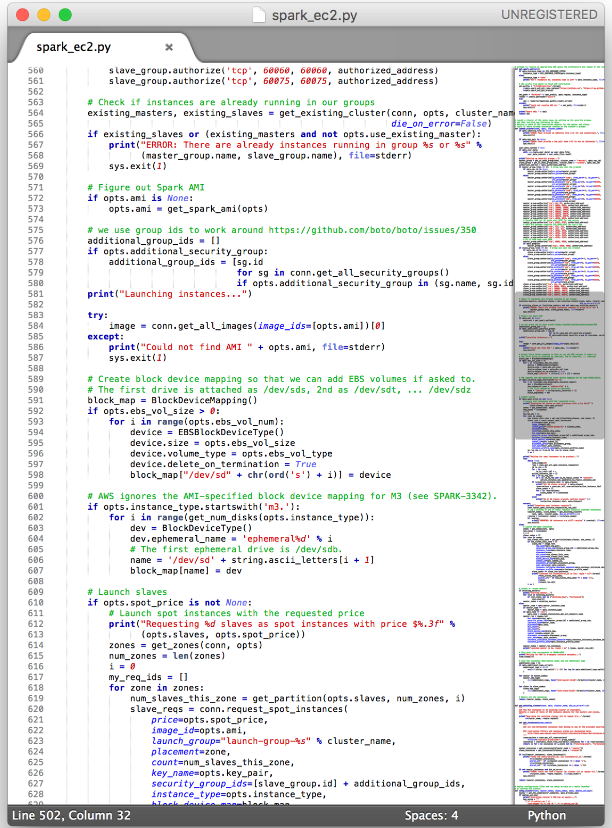
1. Choose a Compute API
2. Choose a Network API
3. Write a Deployment Script

Deployment Script

Simple right?

spark-ec2.py

- Official Spark Script
- 1528 Lines of Code
- Incomprehensible


spark_ec2.py UNREGISTERED

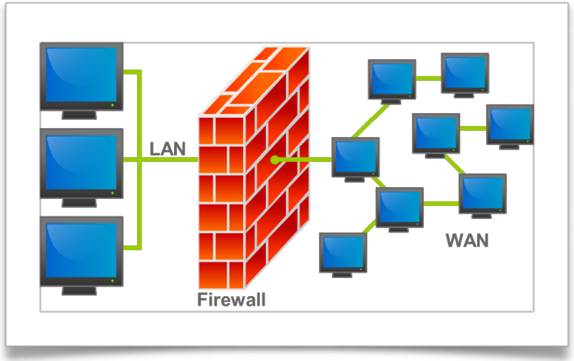
spark_ec2.py

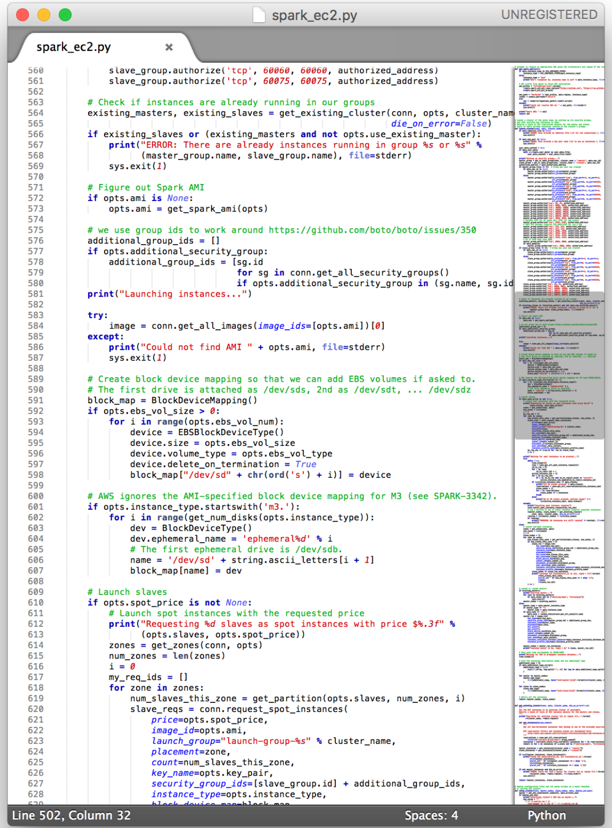
```python
            slave_group.authorize('tcp', 60060, 60060, authorized_address)
            slave_group.authorize('tcp', 60075, 60075, authorized_address)

    # Check if instances are already running in our groups
    existing_masters, existing_slaves = get_existing_cluster(conn, opts, cluster_nam
                                                             die_on_error=False)
    if existing_slaves or (existing_masters and not opts.use_existing_master):
        print("ERROR: There are already instances running in group %s or %s" %
              (master_group.name, slave_group.name), file=stderr)
        sys.exit(1)

    # Figure out Spark AMI
    if opts.ami is None:
        opts.ami = get_spark_ami(opts)

    # we use group ids to work around https://github.com/boto/boto/issues/350
    additional_group_ids = []
    if opts.additional_security_group:
        additional_group_ids = [sg.id
                                for sg in conn.get_all_security_groups()
                                if opts.additional_security_group in (sg.name, sg.id
    print("Launching instances...")

    try:
        image = conn.get_all_images(image_ids=[opts.ami])[0]
    except:
        print("Could not find AMI " + opts.ami, file=stderr)
        sys.exit(1)

    # Create block device mapping so that we can add EBS volumes if asked to.
    # The first drive is attached as /dev/sds, 2nd as /dev/sdt, ... /dev/sdz
    block_map = BlockDeviceMapping()
    if opts.ebs_vol_size > 0:
        for i in range(opts.ebs_vol_num):
            device = EBSBlockDeviceType()
            device.size = opts.ebs_vol_size
            device.volume_type = opts.ebs_vol_type
            device.delete_on_termination = True
            block_map["/dev/sd" + chr(ord('s') + i)] = device

    # AWS ignores the AMI-specified block device mapping for M3 (see SPARK-3342).
    if opts.instance_type.startswith('m3.'):
        for i in range(get_num_disks(opts.instance_type)):
            dev = BlockDeviceType()
            dev.ephemeral_name = 'ephemeral%d' % i
            # The first ephemeral drive is /dev/sdb.
            name = '/dev/sd' + string.ascii_letters[i + 1]
            block_map[name] = dev

    # Launch slaves
    if opts.spot_price is not None:
        # Launch spot instances with the requested price
        print("Requesting %d slaves as spot instances with price $%.3f" %
              (opts.slaves, opts.spot_price))
        zones = get_zones(conn, opts)
        num_zones = len(zones)
        i = 0
        my_req_ids = []
        for zone in zones:
            num_slaves_this_zone = get_partition(opts.slaves, num_zones, i)
            slave_reqs = conn.request_spot_instances(
                price=opts.spot_price,
                image_id=opts.ami,
                launch_group="launch-group-%s" % cluster_name,
                placement=zone,
                count=num_slaves_this_zone,
                key_name=opts.key_pair,
                security_group_ids=[slave_group.id] + additional_group_ids,
                instance_type=opts.instance_type,
                block_device_map=block_map,
```

Line 502, Column 32 Spaces: 4 Python

Network Security

- Status Quo
  - Secure the Perimeter
- A Better Way
  - Tight East-West Firewall
  - Increased script complexity

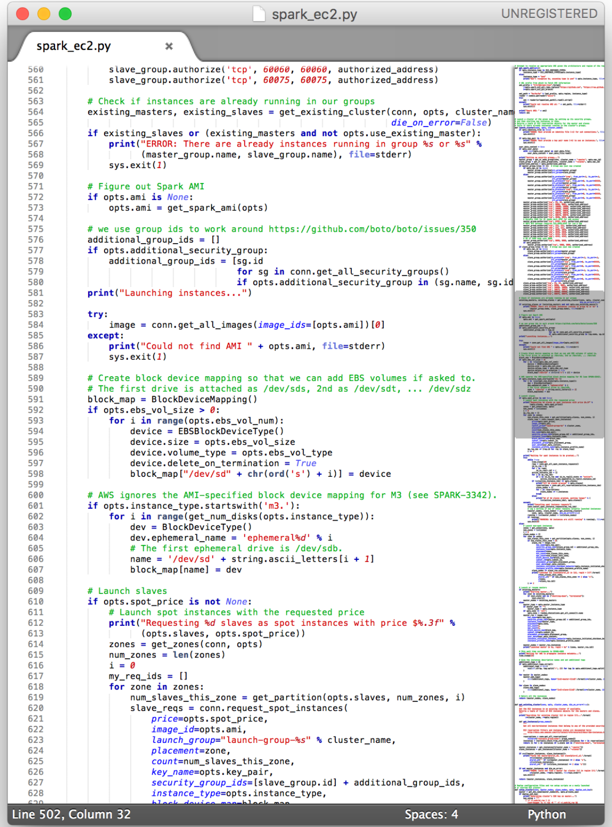




```python
        slave_group.authorize('tcp', 60060, 60060, authorized_address)
        slave_group.authorize('tcp', 60075, 60075, authorized_address)

    # Check if instances are already running in our groups
    existing_masters, existing_slaves = get_existing_cluster(conn, opts, cluster_nam
                                                             die_on_error=False)
    if existing_slaves or (existing_masters and not opts.use_existing_master):
        print("ERROR: There are already instances running in group %s or %s" %
              (master_group.name, slave_group.name), file=stderr)
        sys.exit(1)

    # Figure out Spark AMI
    if opts.ami is None:
        opts.ami = get_spark_ami(opts)

    # we use group ids to work around https://github.com/boto/boto/issues/350
    additional_group_ids = []
    if opts.additional_security_group:
        additional_group_ids = [sg.id
                                for sg in conn.get_all_security_groups()
                                if opts.additional_security_group in (sg.name, sg.id
    print("Launching instances...")

    try:
        image = conn.get_all_images(image_ids=[opts.ami])[0]
    except:
        print("Could not find AMI " + opts.ami, file=stderr)
        sys.exit(1)

    # Create block device mapping so that we can add EBS volumes if asked to.
    # The first drive is attached as /dev/sds, 2nd as /dev/sdt, ... /dev/sdz
    block_map = BlockDeviceMapping()
    if opts.ebs_vol_size > 0:
        for i in range(opts.ebs_vol_num):
            device = EBSBlockDeviceType()
            device.size = opts.ebs_vol_size
            device.volume_type = opts.ebs_vol_type
            device.delete_on_termination = True
            block_map["/dev/sd" + chr(ord('s') + i)] = device

    # AWS ignores the AMI-specified block device mapping for M3 (see SPARK-3342).
    if opts.instance_type.startswith('m3.'):
        for i in range(get_num_disks(opts.instance_type)):
            dev = BlockDeviceType()
            dev.ephemeral_name = 'ephemeral%d' % i
            # The first ephemeral drive is /dev/sdb.
            name = '/dev/sd' + string.ascii_letters[i + 1]
            block_map[name] = dev

    # Launch slaves
    if opts.spot_price is not None:
        # Launch spot instances with the requested price
        print("Requesting %d slaves as spot instances with price $%.3f" %
              (opts.slaves, opts.spot_price))
        zones = get_zones(conn, opts)
        num_zones = len(zones)
        i = 0
        my_req_ids = []
        for zone in zones:
            num_slaves_this_zone = get_partition(opts.slaves, num_zones, i)
            slave_reqs = conn.request_spot_instances(
                price=opts.spot_price,
                image_id=opts.ami,
                launch_group="launch-group-%s" % cluster_name,
                placement=zone,
                count=num_slaves_this_zone,
                key_name=opts.key_pair,
                security_group_ids=[slave_group.id] + additional_group_ids,
                instance_type=opts.instance_type,
```

Portability

amazon web services™
EC2
?
Microsoft Azure



```python
            slave_group.authorize('tcp', 60060, 60060, authorized_address)
            slave_group.authorize('tcp', 60075, 60075, authorized_address)

    # Check if instances are already running in our groups
    existing_masters, existing_slaves = get_existing_cluster(conn, opts, cluster_nam
                                                             die_on_error=False)
    if existing_slaves or (existing_masters and not opts.use_existing_master):
        print("ERROR: There are already instances running in group %s or %s" %
              (master_group.name, slave_group.name), file=stderr)
        sys.exit(1)

    # Figure out Spark AMI
    if opts.ami is None:
        opts.ami = get_spark_ami(opts)

    # we use group ids to work around https://github.com/boto/boto/issues/350
    additional_group_ids = []
    if opts.additional_security_group:
        additional_group_ids = [sg.id
                                for sg in conn.get_all_security_groups()
                                if opts.additional_security_group in (sg.name, sg.id
    print("Launching instances...")

    try:
        image = conn.get_all_images(image_ids=[opts.ami])[0]
    except:
        print("Could not find AMI " + opts.ami, file=stderr)
        sys.exit(1)

    # Create block device mapping so that we can add EBS volumes if asked to.
    # The first drive is attached as /dev/sds, 2nd as /dev/sdt, ... /dev/sdz
    block_map = BlockDeviceMapping()
    if opts.ebs_vol_size > 0:
        for i in range(opts.ebs_vol_num):
            device = EBSBlockDeviceType()
            device.size = opts.ebs_vol_size
            device.volume_type = opts.ebs_vol_type
            device.delete_on_termination = True
            block_map["/dev/sd" + chr(ord('s') + i)] = device

    # AWS ignores the AMI-specified block device mapping for M3 (see SPARK-3342).
    if opts.instance_type.startswith('m3.'):
        for i in range(get_num_disks(opts.instance_type)):
            dev = BlockDeviceType()
            dev.ephemeral_name = 'ephemeral%d' % i
            # The first ephemeral drive is /dev/sdb.
            name = '/dev/sd' + string.ascii_letters[i + 1]
            block_map[name] = dev

    # Launch slaves
    if opts.spot_price is not None:
        # Launch spot instances with the requested price
        print("Requesting %d slaves as spot instances with price $%.3f" %
              (opts.slaves, opts.spot_price))
        zones = get_zones(conn, opts)
        num_zones = len(zones)
        i = 0
        my_req_ids = []
        for zone in zones:
            num_slaves_this_zone = get_partition(opts.slaves, num_zones, i)
            slave_reqs = conn.request_spot_instances(
                price=opts.spot_price,
                image_id=opts.ami,
                launch_group="launch-group-%s" % cluster_name,
                placement=zone,
                count=num_slaves_this_zone,
                key_name=opts.key_pair,
                security_group_ids=[slave_group.id] + additional_group_ids,
                instance_type=opts.instance_type,
```

Line 502, Column 32 Spaces: 4 Python

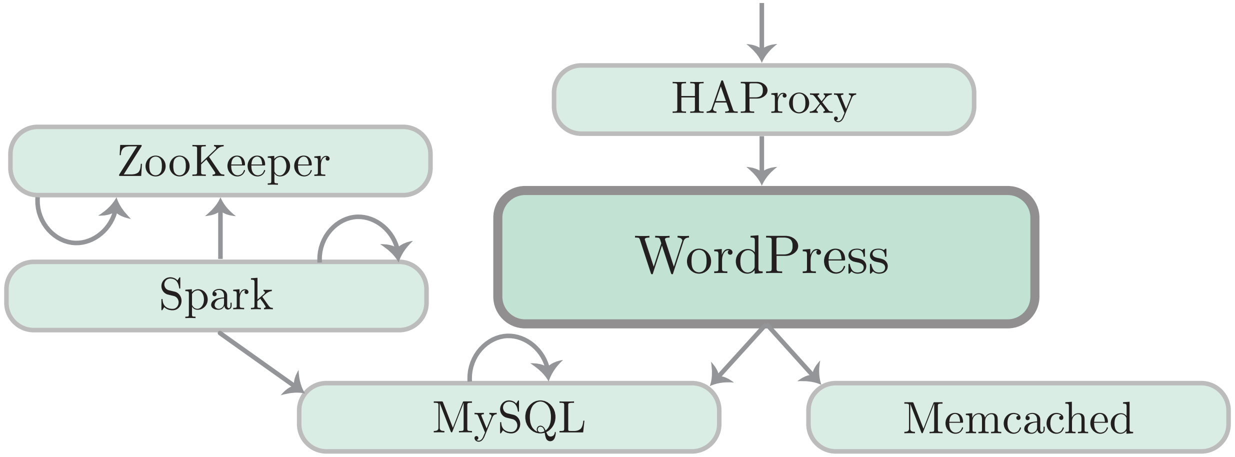
Quilt

Automated Deployment

Quilt DSL: Stitch

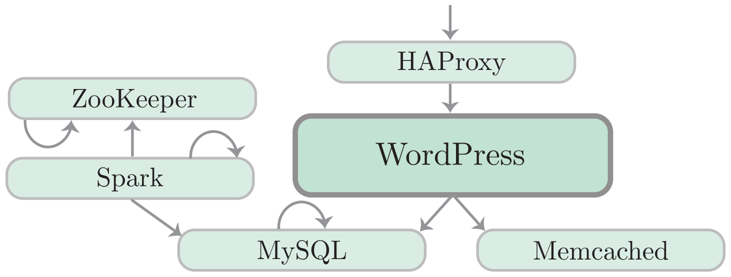
- Declarative Application Specification
- Lisp Dialect
- Declaration Includes:
  - Application Network and Compute
  - Infrastructure

Example: Wordpress



WordPress

```
(import "haproxy")
(import "memcached")
(import "mysql")
(import "spark")
(import "wordpress")
(import "zookeeper")

(let ((db (mysql.New "db" 2))
      (memcd (memcached.New "memcd" 3))
      (wp (wordpress.New "wp" 8 db memcd))
      (hap (haproxy.New "hap" 2 wp))
      (zk (zookeeper.New "zk" 3))
      (spark (spark.New "spark" 2 4 zk)))
  (connect 7077 (hmapValues spark)
                (hmapValues db))
  (connect 80 "public" hap))
```



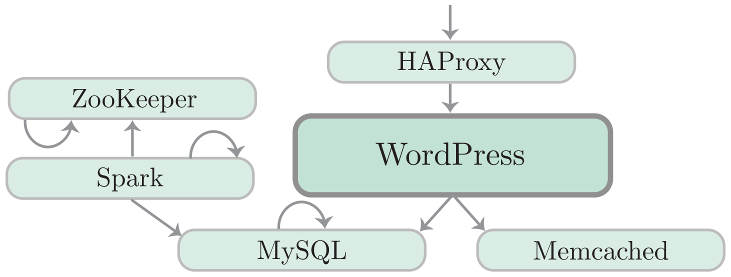
wordpress.New

```
(define (New name n db memcd)
  (let ((dk (makeList n (docker image)))
        (labelNames (strings.Range name n))
        (wp (map label labelNames dk)))
    (configure wp db memcd)
    (connect 3306 wp (hmapGet db "master"))
    (connect 3306 wp (hmapGet db "slave"))
    (connect 11211 wp memcd)
    wp))
```

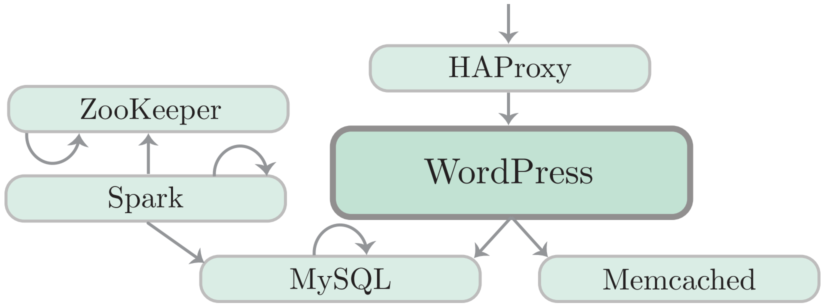
```
(wordpress.New "wp" 8 db memcd)
```

WordPress

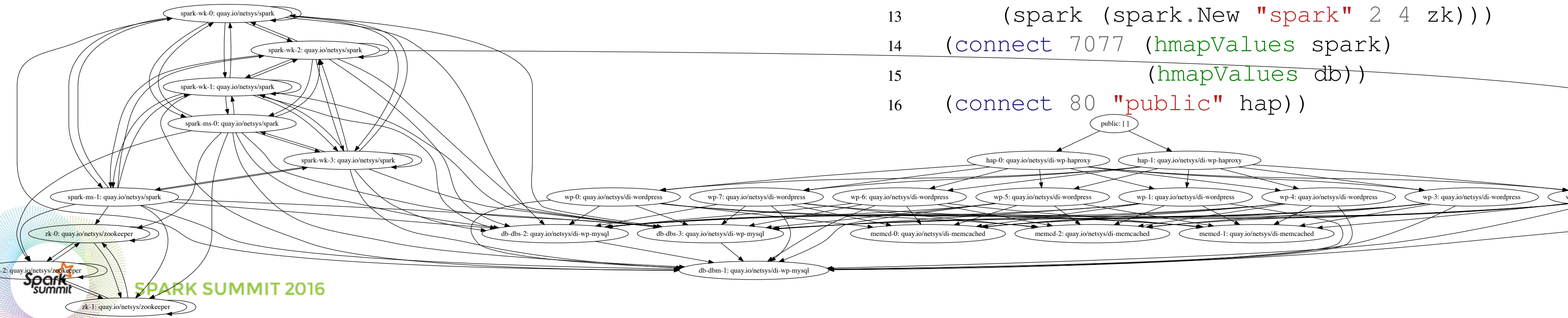
```
(import "haproxy")
(import "memcached")
(import "mysql")
(import "spark")
(import "wordpress")
(import "zookeeper")

(let ((db (mysql.New "db" 2))
      (memcd (memcached.New "memcd" 3))
      (wp (wordpress.New "wp" 8 db memcd))
      (hap (haproxy.New "hap" 2 wp))
      (zk (zookeeper.New "zk" 3))
      (spark (spark.New "spark" 2 4 zk)))
  (connect 7077 (hmapValues spark)
                (hmapValues db))
  (connect 80 "public" hap))
```



WordPress



```
(import "haproxy")
(import "memcached")
(import "mysql")
(import "spark")
(import "wordpress")
(import "zookeeper")

(let ((db (mysql.New "db" 2))
      (memcd (memcached.New "memcd" 3))
      (wp (wordpress.New "wp" 8 db memcd))
      (hap (haproxy.New "hap" 2 wp))
      (zk (zookeeper.New "zk" 3))
      (spark (spark.New "spark" 2 4 zk)))
  (connect 7077 (hmapValues spark)
                (hmapValues db))
  (connect 80 "public" hap))
```


spark-wk-0: quay.io/netsys/spark
spark-wk-2: quay.io/netsys/spark
spark-wk-1: quay.io/netsys/spark
spark-ms-0: quay.io/netsys/spark
spark-wk-3: quay.io/netsys/spark
public: [ ]
hap-0: quay.io/netsys/di-wp-haproxy
hap-1: quay.io/netsys/di-wp-haproxy
spark-ms-1: quay.io/netsys/spark
wp-0: quay.io/netsys/di-wordpress
wp-7: quay.io/netsys/di-wordpress
wp-6: quay.io/netsys/di-wordpress
wp-5: quay.io/netsys/di-wordpress
wp-1: quay.io/netsys/di-wordpress
wp-4: quay.io/netsys/di-wordpress
wp-3: quay.io/netsys/di-wordpress
zk-0: quay.io/netsys/zookeeper
db-dbs-2: quay.io/netsys/di-wp-mysql
db-dbs-3: quay.io/netsys/di-wp-mysql
memcd-0: quay.io/netsys/di-memcached
memcd-2: quay.io/netsys/di-memcached
memcd-1: quay.io/netsys/di-memcached
-2: quay.io/netsys/zookeeper
db-dbm-1: quay.io/netsys/di-wp-mysql
zk-1: quay.io/netsys/zookeeper

Infrastructure

```
(define cfg
  (list (provider "Amazon") (region "us-west-1")
        (ram 32 64) (cpu 4 8) (sshkey "elided")))

(makeList 3 (machine (role "Master") cfg))
(makeList 32 (machine (role "Worker") cfg))
```

Infrastructure

```
(define cfg
  (list (provider "Amazon") (region "us-west-1")
        (ram 32 64) (cpu 4 8) (sshkey "elided")))

(makeList 3 (machine (role "Master") cfg))
(makeList 32 (machine (role "Worker") cfg))
```

amazon web services™ | EC2 → Microsoft Azure ?

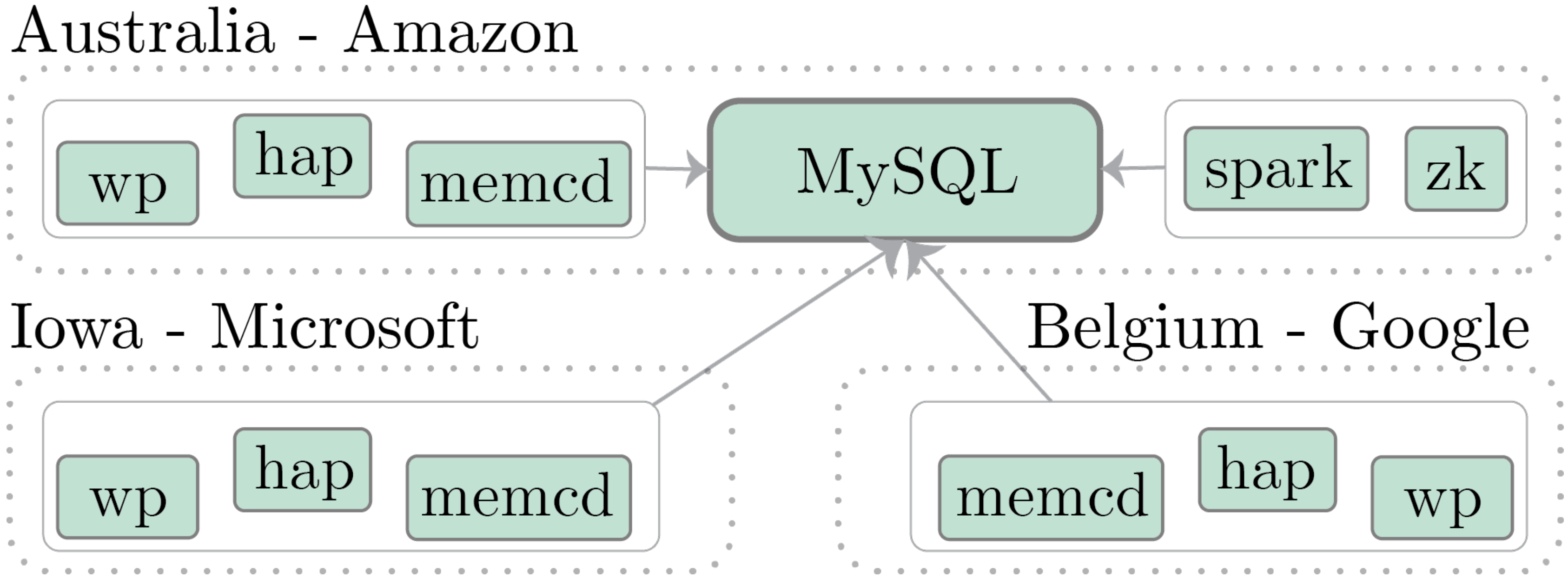
Infrastructure

```
                                  Azure                      Central US
(define cfg
  (list (provider "~~Amazon~~") (region "~~us-west-1~~")
        (ram 32 64) (cpu 4 8) (sshkey "elided")))

(makeList 3 (machine (role "Master") cfg))
(makeList 32 (machine (role "Worker") cfg))
```

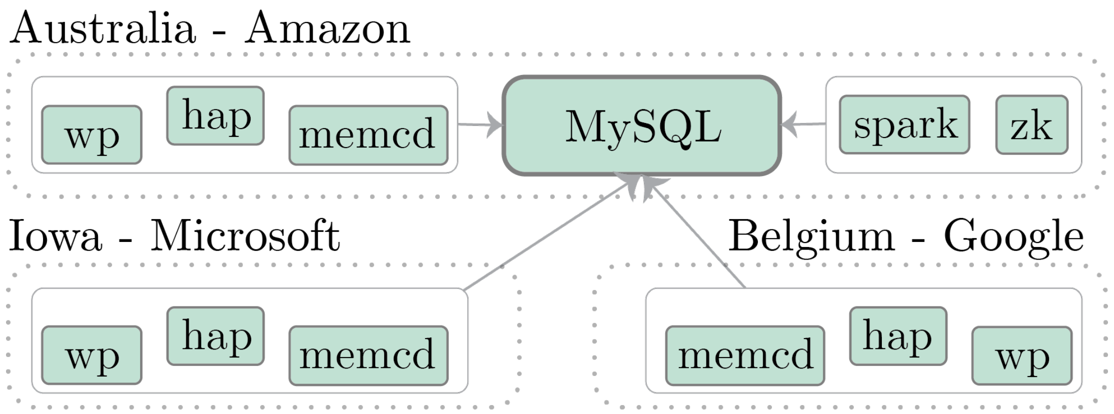
amazon web services™ EC2 → Microsoft Azure ?

SPARK SUMMIT 2016

Geographical Distribution



Geographical Distribution



```
(define cfg (list (ram 32 64) (cpu 4 8)
                  (sshkey "<elided>")))

(define db (mysql.New "db" 2))
(define zk (zookeeper.New "zk" 3))
(define spark (spark.New "spark" 2 4 zk))
(connect 7077 (hmapValues spark) (hmapValues db))

(define (makeLoc prvd rgn)
  (list (provider prvd) (region rgn)))

(define (makePod name)
  (let ((memcd (memcached.New (+ name "-mem") 1))
        (wp (wordpress.New (+ name "-wp")
                           2 db memcd))
        (hap (haproxy.New (+ name "-hap") 1 wp)))
    (connect 80 "public" hap)
    (list memcd wp hap)))

(define (deploy pod loc)
  (makeList 16 (machine (role "Worker") cfg loc))
  (place (machineRule "on" loc) pod))

(deploy (makePod "gce")
        (makeLoc "Google" "europe-west1-b"))

(deploy (makePod "azure")
        (makeLoc "Azure" "Central US"))

(let ((loc (makeLoc "Amazon" "ap-southeast-2"))
      (nodes (append (makePod "aws") zk
                     (hmapValues db)
                     (hmapValues spark))))
    (machine (role "Master") cfg loc)
    (deploy nodes loc))
```

Stitch

Stitch

- Lisp (Scheme)
  - Variables
  - Arithmetic
  - Functions
  - Modules
- Domain Specific Primitives

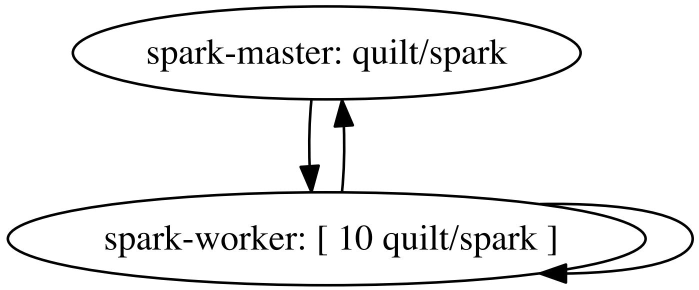
Stitch — Primitives

- Application Primitives
  - “docker”, “label”, “connect”, “place”, “setEnv”
- Infrastructure Primitives
  - “machine”
  - “role”, “provider”, “region”, “ram”, “cpu”, “size”

Stitch — Primitives

```
(label "spark-master" (docker "quilt/spark" "start-master.sh"))

(label "spark-worker"
       (makeList 10 (docker "quilt/spark" "start-worker.sh"
                                          "spark://spark-master.di:7077")))

// Spark workers listen on random ports. Must open up everything.
(connect (list 1000 65535)
         (list "spark-master" "spark-worker")
         "spark-worker")

(connect 7077 "spark-worker" "spark-master")
```

Stitch — Primitives



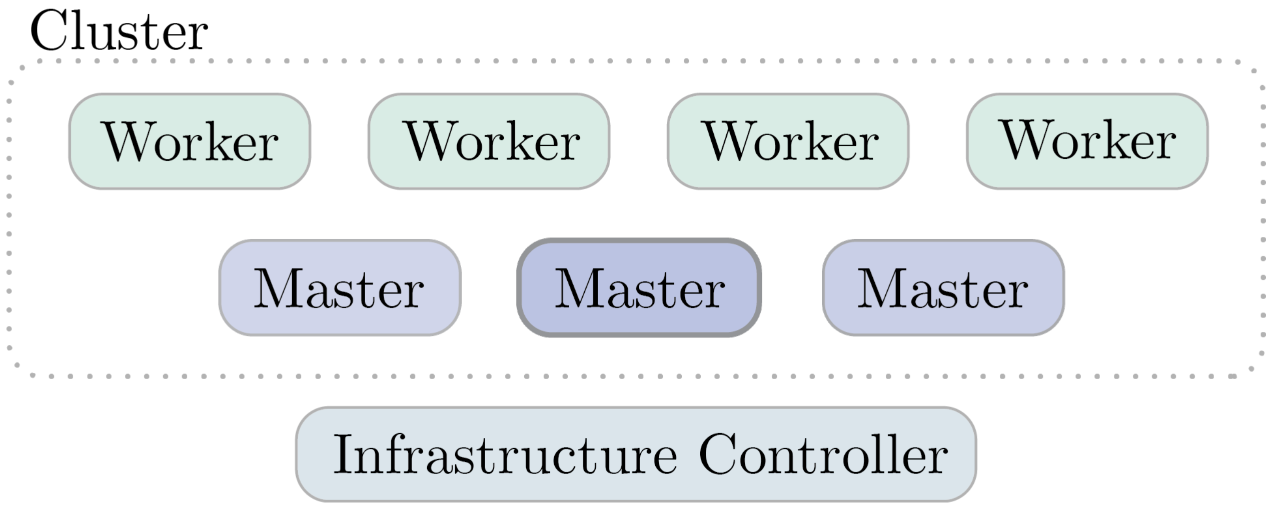
```
(label "spark-master" (docker "quilt/spark" "start-master.sh"))

(label "spark-worker"
       (makeList 10 (docker "quilt/spark" "start-worker.sh"
                            "spark://spark-master.di:7077")))

// Spark workers listen on random ports. Must open up everything.
(connect (list 1000 65535)
         (list "spark-master" "spark-worker")
         "spark-worker")

(connect 7077 "spark-worker" "spark-master")
```

Quilt Architecture

Goals

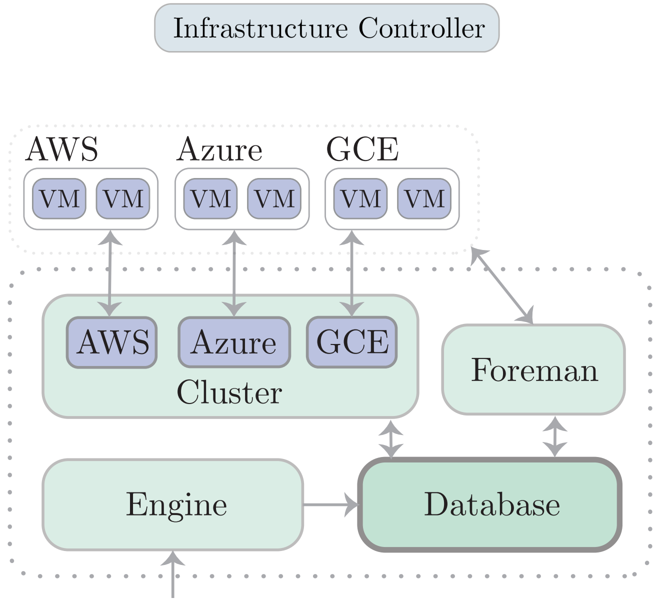
- Simple
- Robust
- Portable

Quilt Architecture



Infrastructure Controller

- Import Infrastructure Spec
- Update Cluster
- Cloud Provider Plugins
  - Amazon EC2
  - Google Compute Engine
  - Microsoft Azure



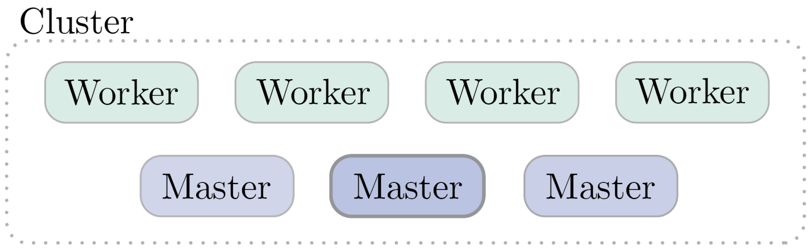
Cloud Provider

- Boot, Stop, List
- Network Reachability
- **Application Agnostic**

```
type Provider interface {
    Connect(namespace string) error

    List() ([]Machine, error)

    Boot([]Machine) error

    Stop([]Machine) error

    SetACLs(acls []string) error

    Disconnect()

    ChooseSize(ram dsl.Range, cpu dsl.Range,
               maxPrice float64) string
}
```

Quilt Cluster

- Virtual Machines Running …
- Application Containers
- Open Virtual Network
  - SDN Overlay
- **Infrastructure Agnostic**



Unsolved Problems

- Application Configuration
- Container Security
- State
- External Services

Related Work

Related Work

- Container Orchestrators
  - Kubernetes, Docker Swarm, Mesos, Nomad
  - No explicit application specification
  - No tight network firewall
- Quilt is a policy layer above these systems

Related Work

- Docker Compose / Kubernetes Helm
  - Declare Groups of Containers to Boot
- Static Data Serialization Format
  - Poor modularity
- Missing network graph

Future Work

Stitch: New Domains

- Security policy
  - Key Management
  - User Management
- Data
- Application Configuration

Stitch Analysis

- Verification
  - Stitch specifies app *entirely*
  - Simpler to verify than deployed systems
- Reachability
- Availability

Summary

- Portable Application Deployment
- Strict Network Security
- Modular, Shareable, Reusable Specifications
- In Future — Formal Analysis

Thank you

quilt.io

ejj@eecs.berkeley.edu