SIMPLIFYING CONTAINERS AT SCALE

THE CONCIERGE PARADIGM
YOUR CONCIERGE FOR THE EVENING

- Gareth Brown
- Director and technologist at Mesoform
- Specialise in securely simplifying and streamlining
- DevOps back in early '00s
- Was running containers in production many years ago
- Built a self-service VM infrastructure..
HISTORY OF CONTAINERS

- 1979: chroot
- Along comes AWS
- Docked back in
FLYING FISH

- Docker Engine in the Cloud
- Maintaining pets
- Monitoring
- Scheduling
- Auto-scaling
- Service discovery
The Concierge Paradigm

Layer Cake

- New technologies (Kubernetes, Mesos)
- Complex
- Integrating different workloads and IaaS
- Up-skilling and support
- Tight coupling and dependency
- Keep It Stupidly Simple
THEY SHOULD REMAKE "BACK TO THE FUTURE 2" WHERE THERE ARE NO FLYING CARS
AND PEOPLE JUST STARE AT THEIR PHONES ALL THE TIME GETTING OFFENDED AT EVERYTHING
FUTURE OF CONTAINERS

- Standardisation
- Portability
- Performance
- Simplified management
- Resource Utilisation
- Cost!
PUPPIES MAKE PAIN GO AWAY
OPERATING PAINS

- On-Premise, EC2, ECS, CoreOS, Kubernetes, other AWS services, Java, Python...
- Papertrail and Elastic Stack
- Zabbix and Librato
- Dropwizard with agents pulling from applications
- bumped all of the common issues
- Windowing and performance
THE CONCIERGE PARADIGM

AUTOPILOT PATTERN

- No complex framework
- Service discovery
- Application orchestration
- Small piece of code to automate common actions
THE CONCIERGE PARADIGM

FLYING ON AUTOPILOT

- Scheduler agnostic
- Most things just work
- App-centric orchestration
- Drastically less management
- Production grade environment, test environment time
- Co-processes!
THE CONCIERGE PARADIGM

BATTERIES INCLUDED

▸ Loose-couple to well defined systems
▸ Automatically register our containers
▸ Automatically discover resources
▸ Self-healing or corrective actions
▸ Interact with legacy applications
▸ Compliance scanning
"postStop": ["zabbix_sender",
    "-c", "/etc/coprocesses/zabbix/zabbix_agentd.conf",
    "--key", "container.state",
    "--value", "0"
]
"tasks": [
    {
        "name": "scheduling_status",
        "command": ["zabbix_sender",
            "-c", "/etc/coprocesses/zabbix/zabbix_agentd.conf",
            "--key", "container.state",
            "--value", "1"],
        "frequency": "10000ms",
        "timeout": "3000ms"
    }
],
"coprocesses": [
    {
        "name": "zabbix_agent",
        "command": ["/usr/sbin/zabbix_agentd", "-fc", "/etc/coprocesses/zabbix/zabbix_agentd.conf"],
        "restarts": 3
    }
]
PUSH VS PULL

- Push method: auto-register but no confidence in instance state
- Pull method: centralised configuration but extra management
- Pull understands load and partitioning
- Windowing
- Processing poor performance
- Can we unify push and pull?
THE CONCIERGE PARADIGM

THE CONCIERGE COURIER

- Two purposes (discovery, delivery)
- Learns metrics
- Picks up metrics
- Delivers them
- Records delivery
- Performance?
THE CONCIERGE PARADIGM

CPU UTILISATION
THE CONCIERGE PARADIGM

LOAD

![CPU load graph 1](image1.png)

![CPU load graph 2](image2.png)
THE CONCIERGE PARADIGM

THE CONCIERGE COURIER

- 3rd party features
- No windowing
- High Performance
- Send to anywhere
- Pull from anywhere
- Monitoring system agnostic
def discover_timers():
    """
    Output Zabbix formatted JSON of keys
    """
    # just for testing purposes, simply open a file with metrics
    with open("/tmp/metrics.json", "r") as metrics_file:
        keys = metrics_file.read()
        keys_json = json.loads(keys)

    discovery_data_dict = {
        'data': [{'#TIMER': key} for key in keys_json['timers']]
    }
    print(json.dumps(discovery_data_dict))
def get_timers():
    with open("/tmp/metrics.json", "r") as metrics_file:
        keys = metrics_file.read()
        keys = json.loads(keys)
    with open("/tmp/timer_metrics_zabbix.sender", "w") as sender_file:
        for timer_name, metrics in keys["timers"].items():
            for metric_name, metric_value in metrics.items():
                sender_file.write("- timer[{0}.{1}] {2}\n" .format(timer_name, metric_name, metric_value))
    send_metrics("timer")

def send_metrics(metric_type):
    filename = "/tmp/" + metric_type + "_metrics_zabbix.sender"
    call("zabbix_sender -c /etc/coprocesses/zabbix/zabbix_agentd.conf -i "
        + filename + " >/dev/null", shell=True)
print time.time() - startTime
THE CONCIERGE PARADIGM

THE ENFIELD METHOD

- Accurate, single-shot, immediate feedback
- Like the rifle
- Backoff under network issues
- Greater confidence in container state
- Greater confidence in state of whole system
- More frequent updates
THE CONCIERGE PARADIGM

STATE TO STATE

- State in service discovery
- State in event management
- End-to-end view of whole system
- State history
- Dev/Ops on the same page
- State manipulation!
STATE CONTROL

- Consul keeps configuration state
- Monitoring performance and availability state
- Dynamic Asset database
- Automate scheduling, scaling, archiving
WHAT IF I TOLD YOU
SIRI IS A PERSONAL ASSISTANT, NOT A TALK-BUDDY
THE CONCIERGE PARADIGM

THE CONCIERGE SCHEDULER

- Containers Auto-register
- Push & pull state
- Optimised over many years
- Monitoring grouping them services
- Data about whole system
- Basically just runs `docker-compose scale`
SCALING

- Complex trigger profiles
- Pre-scaling using a predictive trigger
- Compare upstream service performance as well
- Vertical scaling
- Escalation steps
- Scaling events and problem events in one system
# Variable assignment
action=$1; service_name=$2; current_scale=$3; increment=$4

scale_service(){
    /usr/bin/docker-compose --tlsverify --tlscert=${DOCKER_CERT_PATH}cert.pem \
    --tlscacert=${DOCKER_CERT_PATH}ca.pem \
    --tlskey=${DOCKER_CERT_PATH}key.pem --project-name dockerlx \
    --host tcp://dockerapi-private-lab1.mesoform.com:2376 --file /tmp/docker-compose.yml \
    scale ${service_name}=${action}
    echo "$(date): Scaled ${service_name} from ${current_scale} to $1" \
        >> /tmp/app_scheduler_output
    exit 0
}

scale_up(){
    desired_scale=$((current_scale + increment))
    scale_service ${desired_scale}
}

scale_down(){
    desired_scale=$((current_scale - increment))
    scale_service ${desired_scale}
}
# THE CONCIERGE PARADIGM

## MANIFEST COLLECTION

<table>
<thead>
<tr>
<th>Hosts</th>
<th>Key</th>
<th>Interval</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototypes</td>
<td>host_prototypes: system.run[/usr/local/bin/concierge_courier.py]</td>
<td>5m</td>
<td>Zabbix agent (active)</td>
</tr>
<tr>
<td>Prototypes</td>
<td>host_prototypes: vfs.fs.discovery</td>
<td>1h</td>
<td>Zabbix agent (active)</td>
</tr>
<tr>
<td>Prototypes</td>
<td>host_prototypes: net.if.discovery</td>
<td>1h</td>
<td>Zabbix agent (active)</td>
</tr>
</tbody>
</table>
```python
def discover_timers():
    """
    Output Zabbix formatted JSON of keys
    """
    # just for testing purposes, simply open a file with metrics
    with open("/tmp/metrics.json", "r") as metrics_file:
        keys = metrics_file.read()
        keys_json = json.loads(keys)

    discovery_data_dict = {
        'data': [{"#{TIMER}": key} for key in keys_json['timers']]
    }
    print(json.dumps(discovery_data_dict))
```
## Delivery Addresses

<table>
<thead>
<tr>
<th>Key</th>
<th>Interval</th>
<th>History</th>
<th>Trends</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer[#{TIMER}.count]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.max]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.mean]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.mean_rate]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.min]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.p75]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.p95]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.p98]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.p99]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.p999]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
<tr>
<td>timer[#{TIMER}.stddev]</td>
<td>7d</td>
<td>365d</td>
<td>365d</td>
<td>Zabbix trapper</td>
</tr>
</tbody>
</table>
## DELIVERY ADDRESSES

<table>
<thead>
<tr>
<th>OS: Linux</th>
<th>Maximum number of processes</th>
<th>Triggers</th>
<th>kernel.maxproc</th>
<th>1h</th>
<th>7d</th>
<th>365d</th>
<th>Zabbix agent (active)</th>
<th>My App</th>
</tr>
</thead>
<tbody>
<tr>
<td>discover application timers: my.test-timer-0:count</td>
<td>timers[my.test-timer-0.count]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:max</td>
<td>timers[my.test-timer-0.max]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:mean</td>
<td>timers[my.test-timer-0.mean]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:mean_rate</td>
<td>timers[my.test-timer-0.mean_rate]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:min</td>
<td>timers[my.test-timer-0.min]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:p75</td>
<td>timers[my.test-timer-0.p75]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:p95</td>
<td>timers[my.test-timer-0.p95]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:p98</td>
<td>timers[my.test-timer-0.p98]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:p99</td>
<td>timers[my.test-timer-0.p99]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:p999</td>
<td>timers[my.test-timer-0.p999]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-0:stdddev</td>
<td>timers[my.test-timer-0.stdddev]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-10:count</td>
<td>timers[my.test-timer-10.count]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-10:max</td>
<td>timers[my.test-timer-10.max]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-10:mean</td>
<td>timers[my.test-timer-10.mean]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-10:mean_rate</td>
<td>timers[my.test-timer-10.mean_rate]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discover application timers: my.test-timer-10:min</td>
<td>timers[my.test-timer-10.min]</td>
<td>7d</td>
<td>365d</td>
<td>Zabbix trapper</td>
<td>My App</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# The Concierge Paradigm

## Item Delivery

<table>
<thead>
<tr>
<th>Triggers</th>
<th>Key</th>
<th>Interval</th>
<th>History</th>
<th>Trends</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>agent.ping</td>
<td>1m</td>
<td>7d</td>
<td>365d</td>
</tr>
<tr>
<td>2</td>
<td>system.run[/usr/local/bin/concierge_courier.py get_timers <a href="http://localhost:8080/metrics">http://localhost:8080/metrics</a>]</td>
<td>1m</td>
<td>7d</td>
<td>365d</td>
</tr>
<tr>
<td>3</td>
<td>container.state</td>
<td>3d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
def get_timers():
    with open("/tmp/metrics.json", "r") as metrics_file:
        keys = metrics_file.read()
        keys = json.loads(keys)
    with open("/tmp/timer_metrics_zabbix.sender", "w") as sender_file:
        for timer_name, metrics in keys['timers'].items():
            for metric_name, metric_value in metrics.items():
                sender_file.write("- timer[{0}.{1}] {2}\n" .format(timer_name, metric_name, metric_value))
    send_metrics("timer")

def send_metrics(metric_type):
    filename = "/tmp/" + metric_type + "_metrics_zabbix.sender"
    call("zabbix_sender -c /etc/coprocesses/zabbix/zabbix_agentd.conf -i " + filename + " >/dev/null", shell=True)
    print time.time() - startTime
## THE CONCIERGE PARADIGM

### EVENT TRIGGERS

<table>
<thead>
<tr>
<th>Event</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host name has changed</td>
<td><code>{app-my_app:vfs.file_cksum[/etc/passwd].diff}(0)&gt;0</code></td>
</tr>
<tr>
<td>COURIER::metric::collection::FAILED (CODE=ITEM.VALUE)</td>
<td><code>{app-my_app:system.run[/usr/local/bin/concierge_courier.py get_timers http://localhost:8080/metrics].last}&lt;=0</code></td>
</tr>
<tr>
<td>COURIER::metric::collection::$COURIER_SLOW (TIME=ITEM.VALUE)</td>
<td><code>{app-my_app:system.run[/usr/local/bin/concierge_courier.py get_timers http://localhost:8080/metrics].last}&gt;{$COURIER_SLOW}</code></td>
</tr>
<tr>
<td>Container state of (HOST.NAME) is (ITEM.VALUE) is loaded</td>
<td><code>{app-my_app:container.state.last}=0</code></td>
</tr>
<tr>
<td>Swap space on (HOST.NAME) is below 50%</td>
<td><code>{app-my_app:system.swap.size[pfree].last(0)}&lt;50</code></td>
</tr>
<tr>
<td>No heartbeat from (HOST.NAME) in last 2 minutes</td>
<td><code>{app-my_app:container.state.nodata}(2m)=2</code></td>
</tr>
<tr>
<td>CPU load is too high on (HOST.NAME)</td>
<td><code>{app-my_app:system.cpu.load[percpu,avg1].avg}(5m)&gt;5</code></td>
</tr>
<tr>
<td>Push and Pull checks on (HOST.NAME) are both unreachable for 1 minute</td>
<td><code>{app-my_app:agent.ping.nodata}(1m)=1</code> and <code>{app-my_app:container.state.nodata}(1m)=1</code></td>
</tr>
</tbody>
</table>
## Service State

<table>
<thead>
<tr>
<th>Triggers</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>e: Service::Consul::containers::running</td>
<td>Grpsum[&quot;${SERVICE_HOSTGROUP}&quot;],&quot;container.state&quot;,last</td>
</tr>
<tr>
<td>e: Service::Consul::my.test-timer-14::p95</td>
<td>Grpavg[&quot;${SERVICE_HOSTGROUP}&quot;],&quot;timer[my.test-timer-14.p95]&quot;,last</td>
</tr>
<tr>
<td>e: Service::Consul::storage::usage</td>
<td>Grpsum[&quot;${SERVICE_HOSTGROUP}&quot;],&quot;vfs.fs.size[/,free]&quot;,last</td>
</tr>
<tr>
<td>e: Service::Consul::system::load::1min</td>
<td>Grpavg[&quot;${SERVICE_HOSTGROUP}&quot;],&quot;system.cpu.load[percpu,avg1]&quot;,last</td>
</tr>
<tr>
<td>e: Service::Zabbix::agents::responding</td>
<td>Grpsum[&quot;${SERVICE_HOSTGROUP}&quot;],&quot;agent.ping&quot;,last</td>
</tr>
</tbody>
</table>
THE CONCIERGE PARADIGM

SCALING ACTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service scale down due to low load</td>
<td>Maintenance status not in <code>maintenance</code></td>
</tr>
<tr>
<td></td>
<td>Trigger = <code>SERVICES::aggregate: service-aggregates::load::low::*UNKNOWN*</code></td>
</tr>
<tr>
<td>Service scale up due to high load</td>
<td>Maintenance status not in <code>maintenance</code></td>
</tr>
<tr>
<td></td>
<td>Trigger = <code>SERVICES::aggregate: service-aggregates::load::high::*UNKNOWN*</code></td>
</tr>
</tbody>
</table>

Type: Custom script

**Execute on:** Zabbix agent, Zabbix server

**Commands:**
```
/usr/lib/zabbix/externalscripts/concierge_scheduler/concierge_scheduler.sh scale_up {HOST.HOST} {INVENTORY.DEPLOYMENT.STATUS} 1
```
# Variable assignment
action=$1; service_name=$2; current_scale=$3; increment=$4

scale_service(){
    /usr/bin/docker-compose --tlsverify --tlscert=${DOCKER_CERT_PATH}cert.pem \
    --tlscacert=${DOCKER_CERT_PATH}ca.pem \
    --tlskey=${DOCKER_CERT_PATH}key.pem --project-name dockerlx \
    --host tcp://dockerapi-private-lab1.mesoform.com:2376 --file /tmp/docker-compose.yml \
    scale ${service_name}=${1}
    echo "$(date): Scaled ${service_name} from ${current_scale} to $1" \
    >> /tmp/app_scheduler_output
    exit 0
}

scale_up(){
    desired_scale=$((current_scale + increment))
    scale_service ${desired_scale}
}

scale_down(){
    desired_scale=$((current_scale - increment))
    scale_service ${desired_scale}
}
<table>
<thead>
<tr>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ consul.grpavg('${SERVICE_HOSTGROUP}', 'system.cpu.load[percpu,avg1]', last).last() }</code></td>
</tr>
<tr>
<td><code>{ consul.grpavg('${SERVICE_HOSTGROUP}', 'system.cpu.load[percpu,avg1]', last).last() }</code></td>
</tr>
<tr>
<td><code>{ consul.grpsum('${SERVICE_HOSTGROUP}', 'container.state', last).diff() }=1</code></td>
</tr>
</tbody>
</table>
CONCLUSION

- Autopilot Pattern and Enfield Method
- We're already: doing event management, auto-registering, aggregating metrics, performing actions on triggers, maintaining system state, highly optimised, self-healing,
- Controlling the state
- Accuracy and performance
- Short lead time
THE CONCIERGE PARADIGM

WHAT'S NEXT

- Load testing Zabbix Server/Proxy
- Use Zabbix Python interpreter module
- Make this native in Zabbix?
- DevOps everything!
SO LONG AND THANKS FOR ALL THE FISH

- Read the full article at http://www.mesoform.com/blog-listing/info/the-concierge-paradigm
- Search: "mesoform concierge paradigm"
- @MesoformLtd
- /mesoform
- /mesoform
- http://www.mesoform.com/contact-us