AFW: Automating host-based firewalls with Chef

Julien Vehent
Aweber Communications
9th Netfilter Workshop
Open Source Days 2013

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

Problem

Monolithic/border firewalls will either fail under load, or contain too many rules to secure anything.

Solution

Host-based firewalls
and automated rule management.

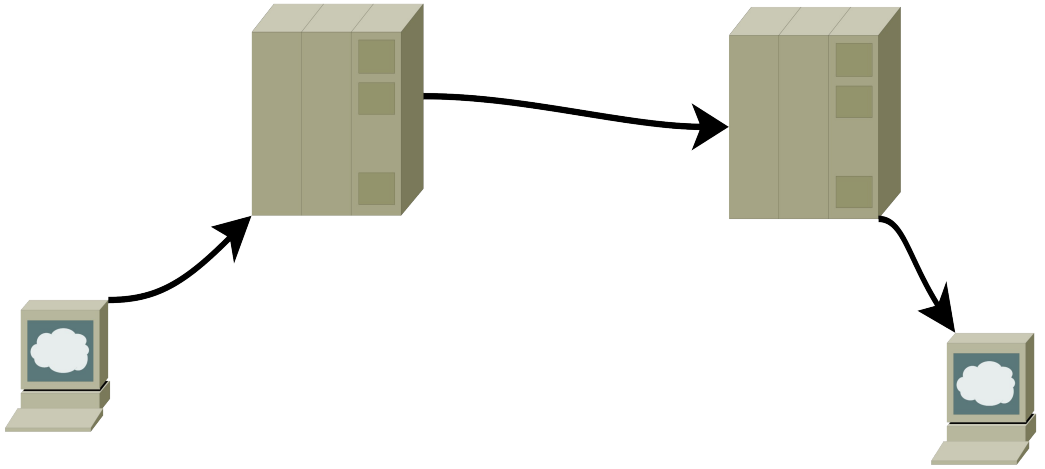
AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

Julien Vehent - @jvehent

- Systems & Security Engineer at AWeber COMMUNICATIONS
- I built and secure web infrastructures on Linux
- http://www.github.com/jvehent/
- http://jve.linuxwall.info

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

The 70's Firewall design

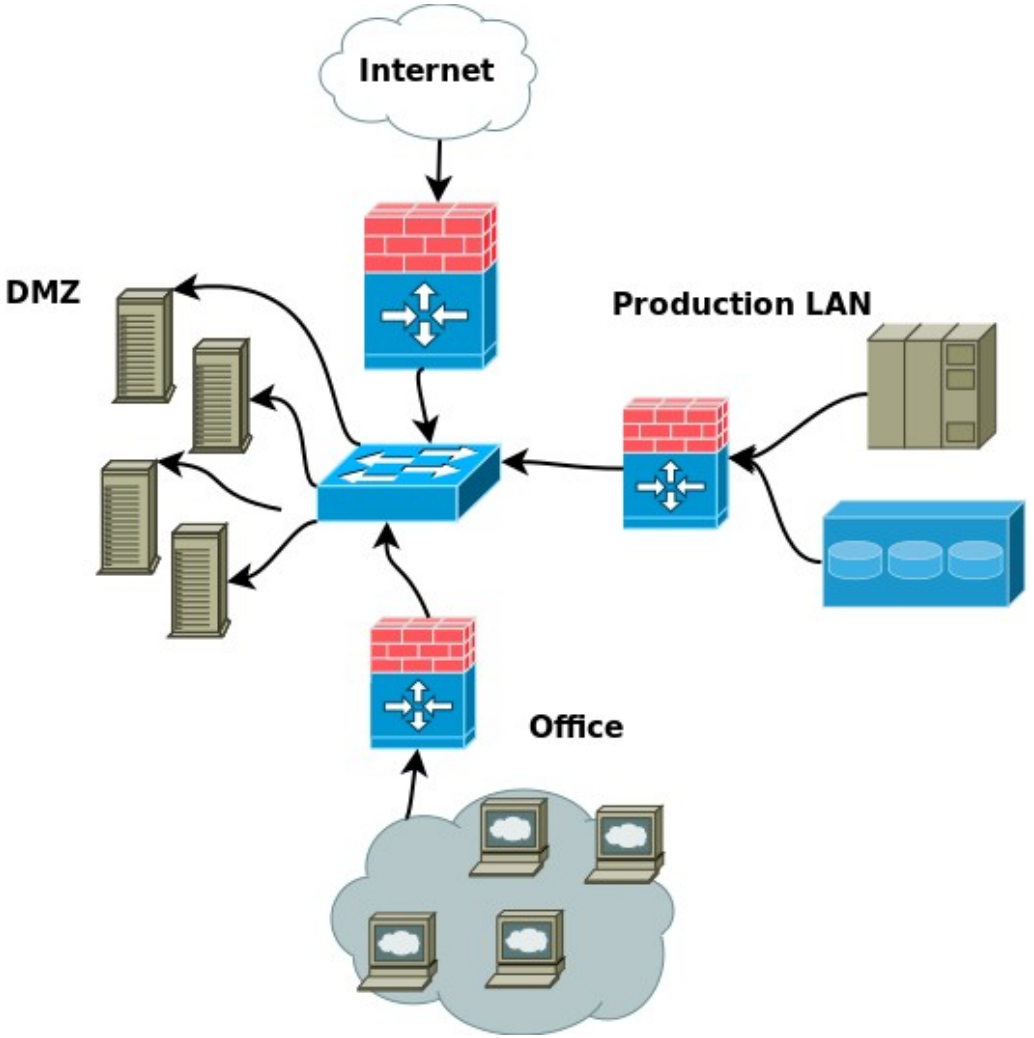


ping ?

WHHAAAA !!!!
I GOT A PING !!!!

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

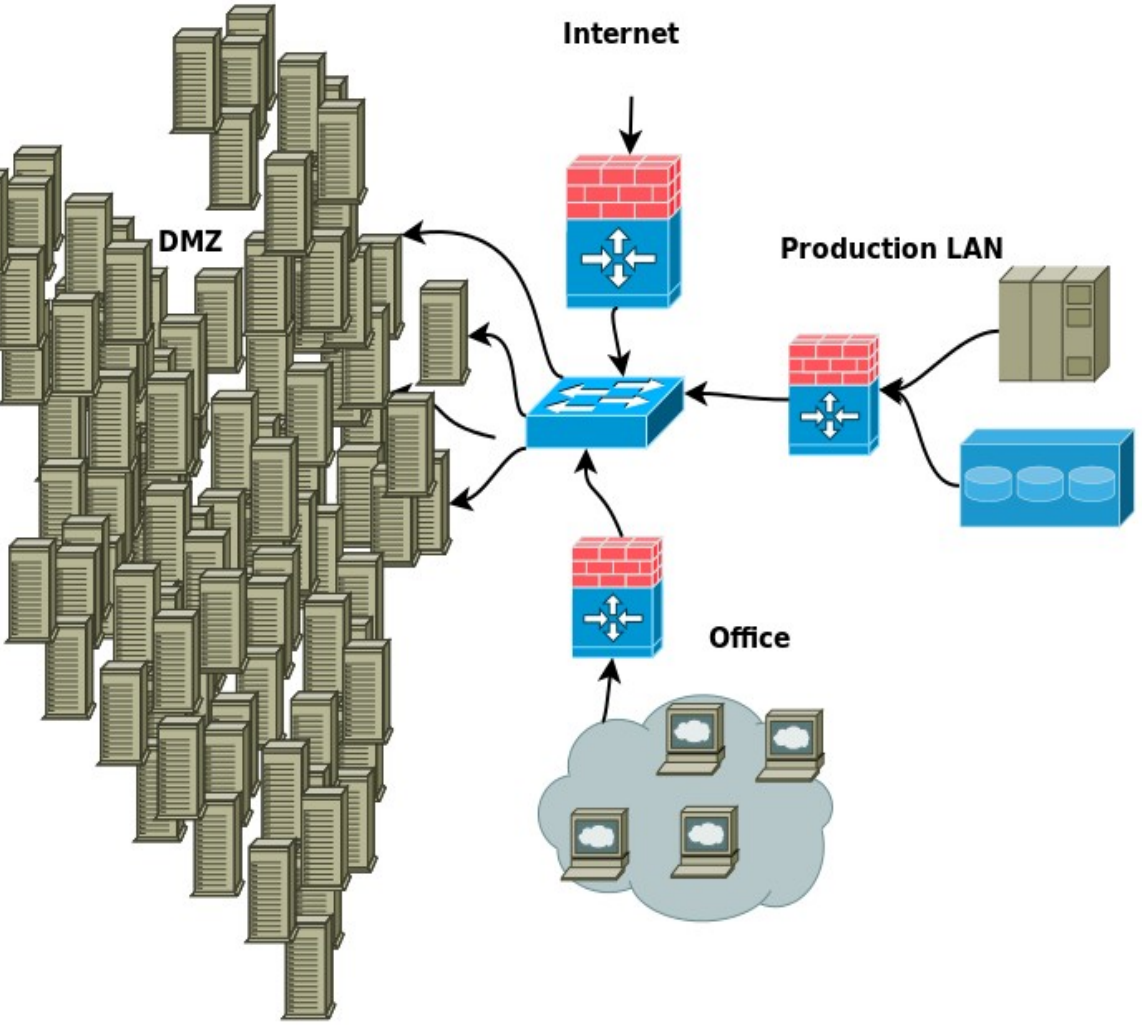
The 90's Firewall design



- Few powerful & expensive firewalls filter the entire traffic
- DMZ design: works with small DMZs
- Rules maintained manually: need a route opening/closing workflow

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

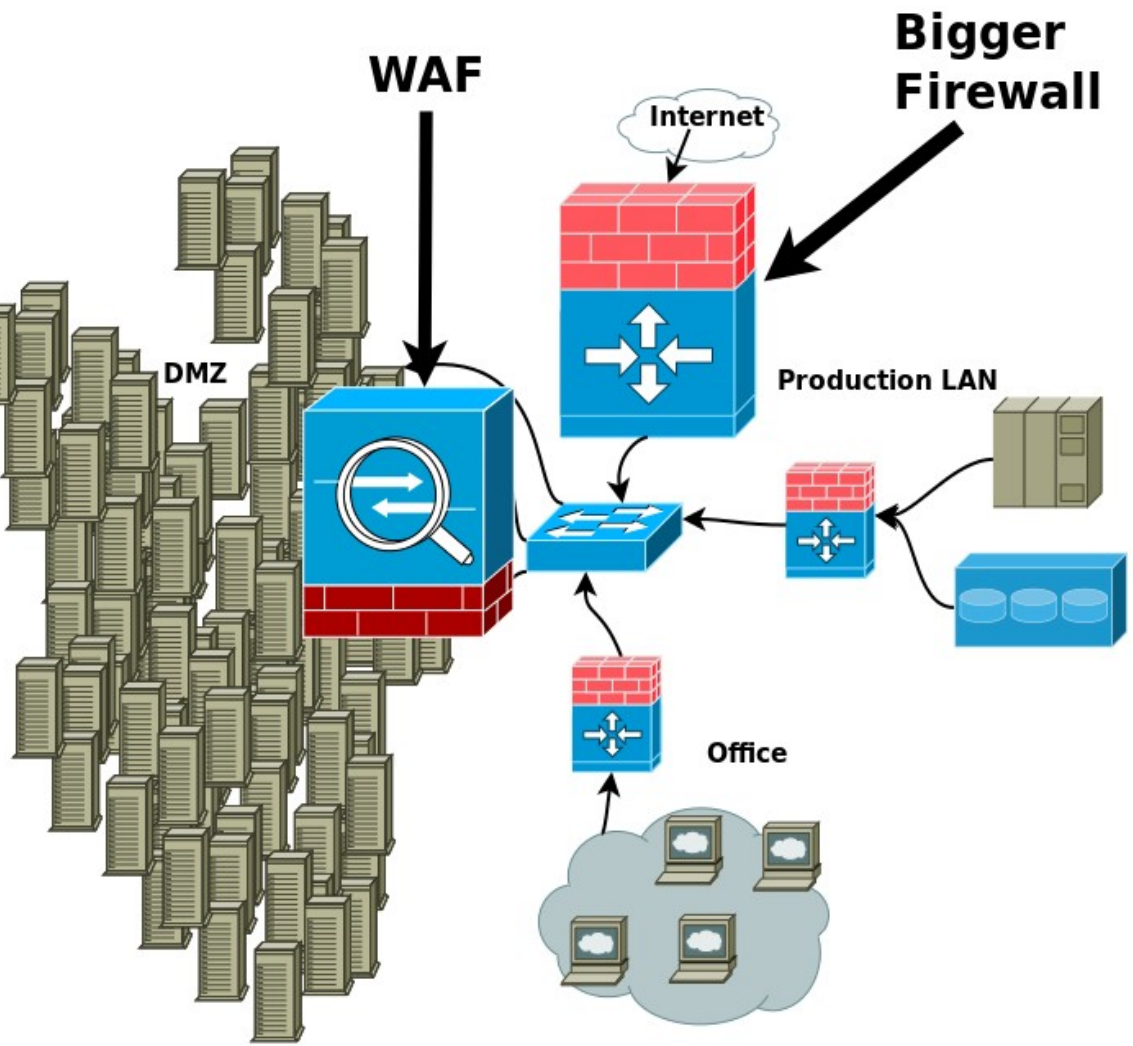
2000 Firewall ~~design~~ failure



- “Let's keep pilling stuff in the DMZ, it needs to be accessed from the Internet anyway.”
- Really complex rule opening workflow
- Rules are closed when someone happens to look at the firewall at 4am on a sunday morning.

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

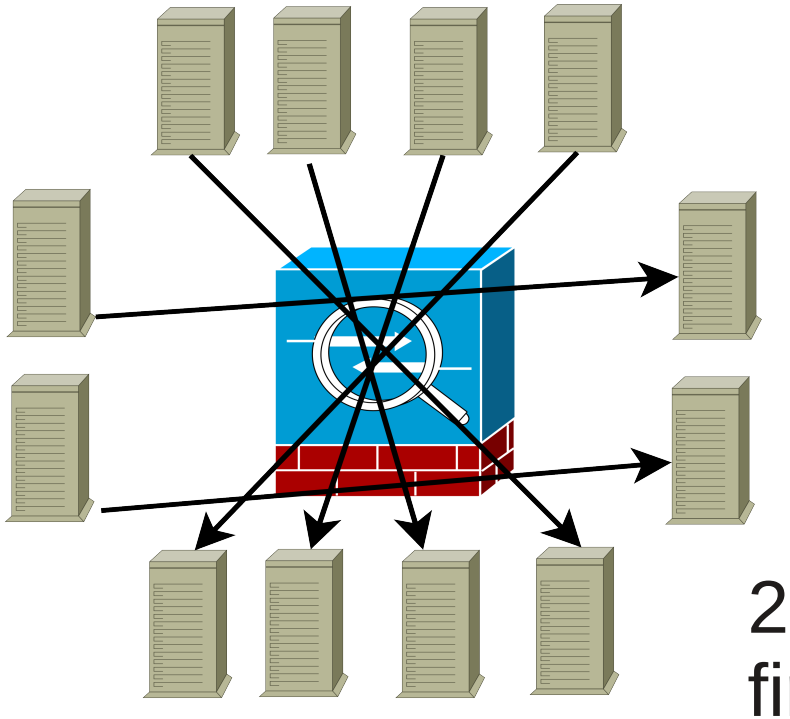
2005: We Need More !



- **Bigger** firewalls
- NIDS, NIPS, HIDS
- Web App Firewall, Database firewalls
- Logs centralizers, Logs analyzers, Logs readers
- Developers stopped trying to connect application A to server B somewhere around 2008

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

2010: Congratulations !



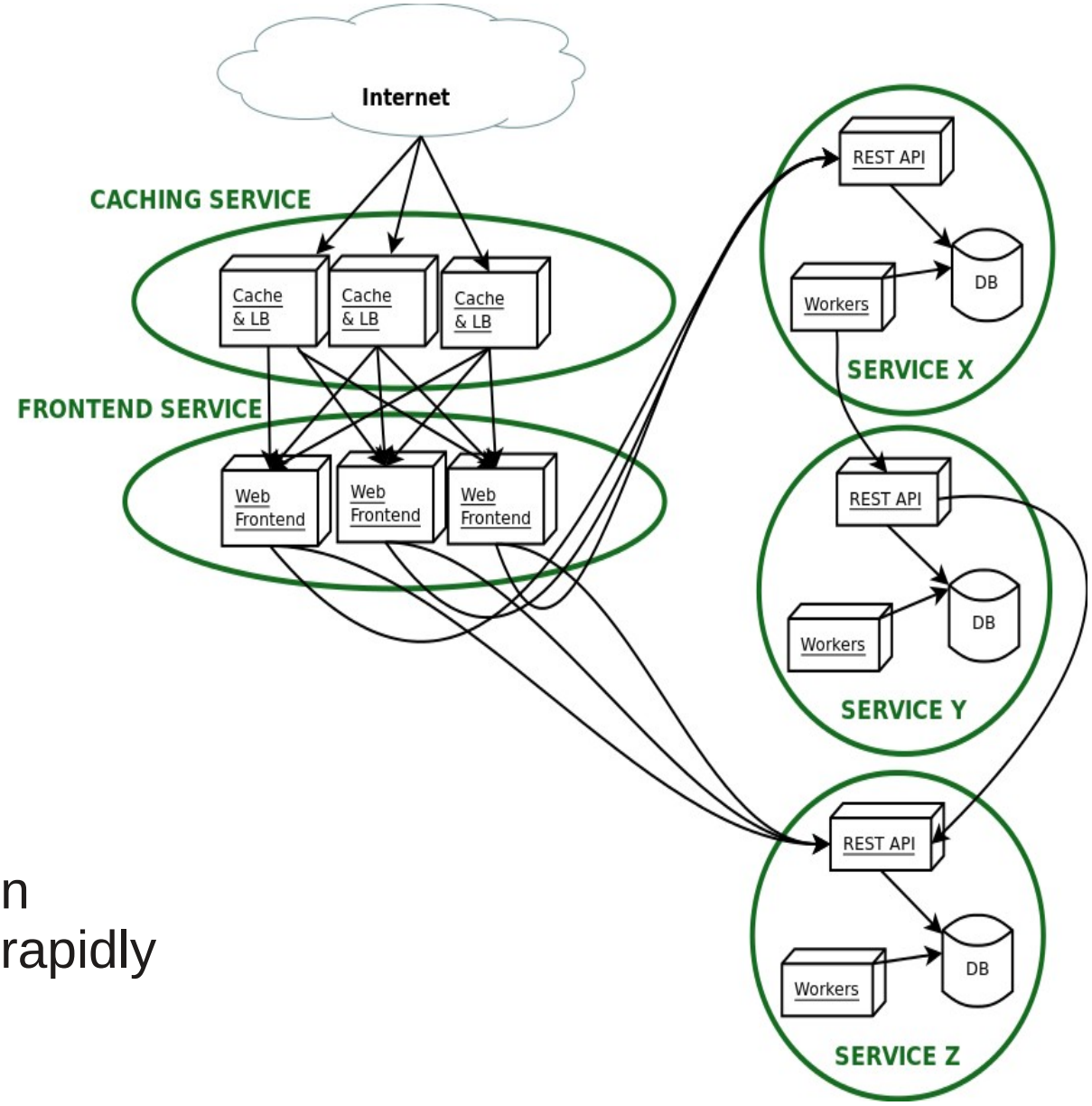
You are now routing your entire datacenter traffic through a handful of appliances from very happy vendors.

25,000 IDS alerts per day, 6GB of firewall logs, added 300ms of latency everywhere... sounds familiar ?

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

Service Oriented Architecture

- Services are autonomous
- Services call each other using a standard protocol (REST: JSON over HTTP)
- The architecture is described by a list of dependencies between services



"Cloud" type requirements:

- No single point of failure
- Optimize resources utilization
- Augment & reduce capacity rapidly

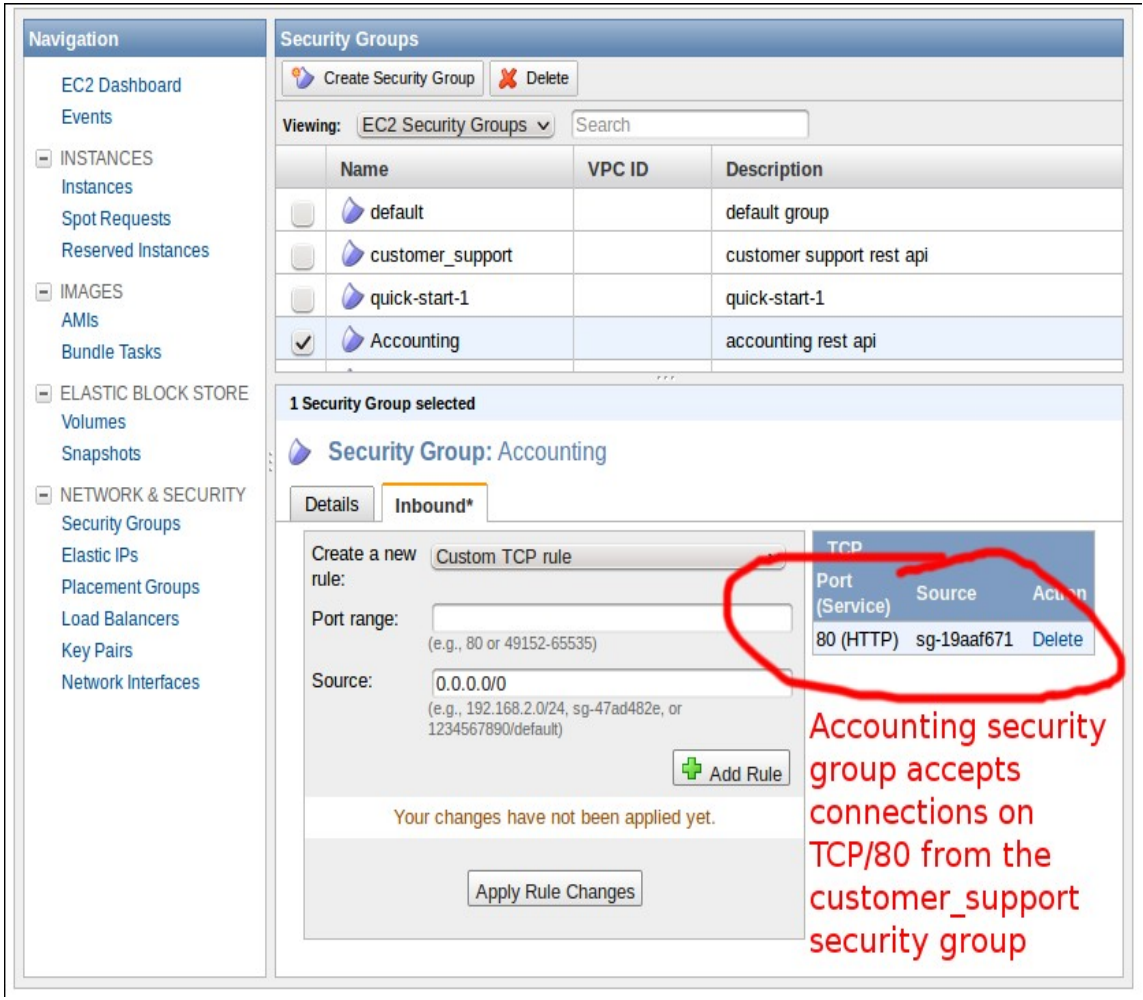
AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

Service Oriented Security

AWS security groups (SG)

- Create SG-X for service X
- Create SG-Y for service Y
- Allow SG-X to connect to SG-Y
- All instances (servers) in SG-X will be allowed to connect to SG-Y

=> Dynamic security: No need to update the firewall for each new server



AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

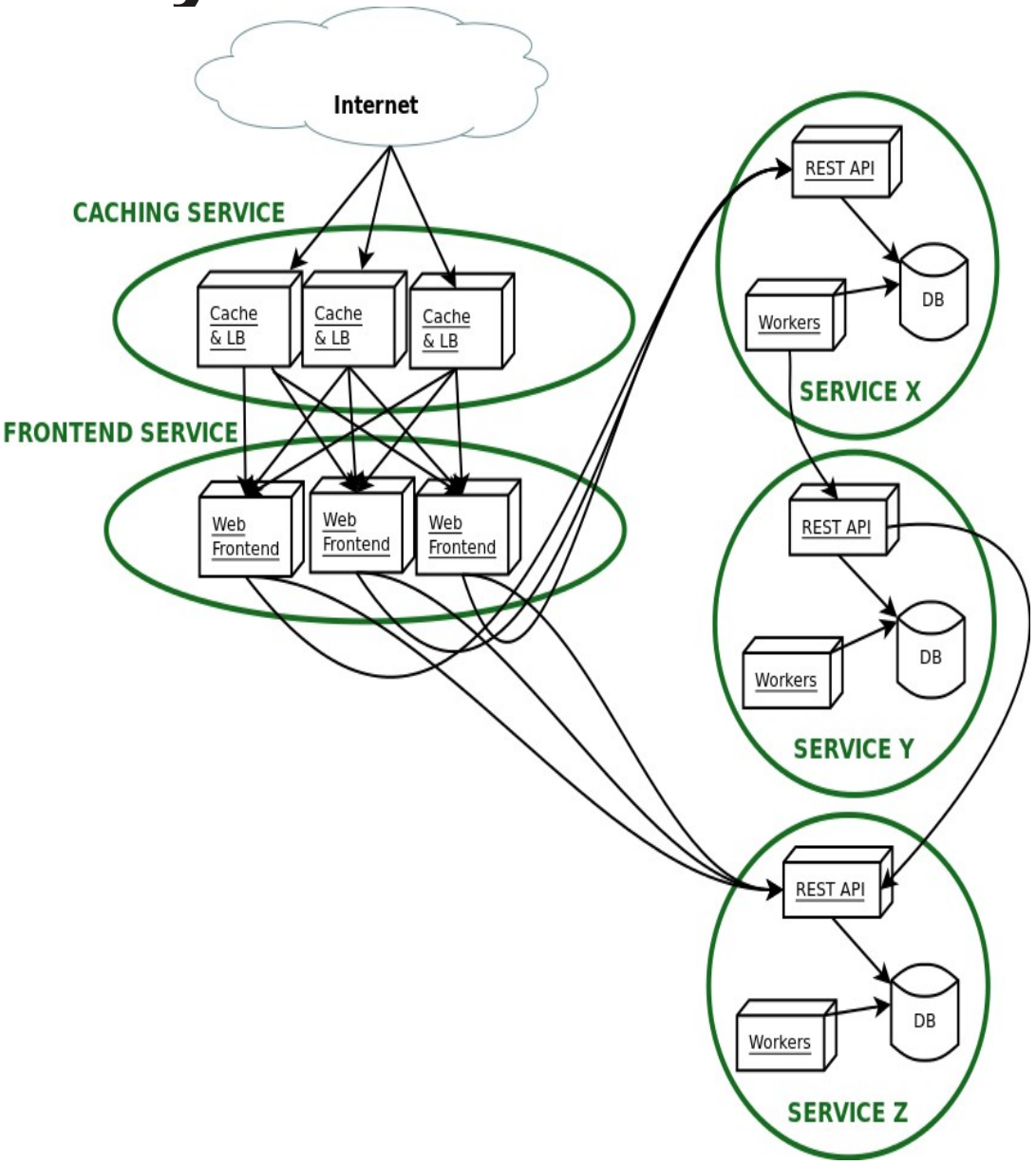
Service Oriented Security

Inter-services policy

```
ACCEPT 0.0.0.0/0 to CACHING on TCP/80

ACCEPT CACHING to FRONTEND on TCP/80

ACCEPT FRONTEND to ServiceX on TCP/80

ACCEPT FRONTEND to ServiceZ on TCP/80

ACCEPT ServiceX to ServiceY on TCP/80
```

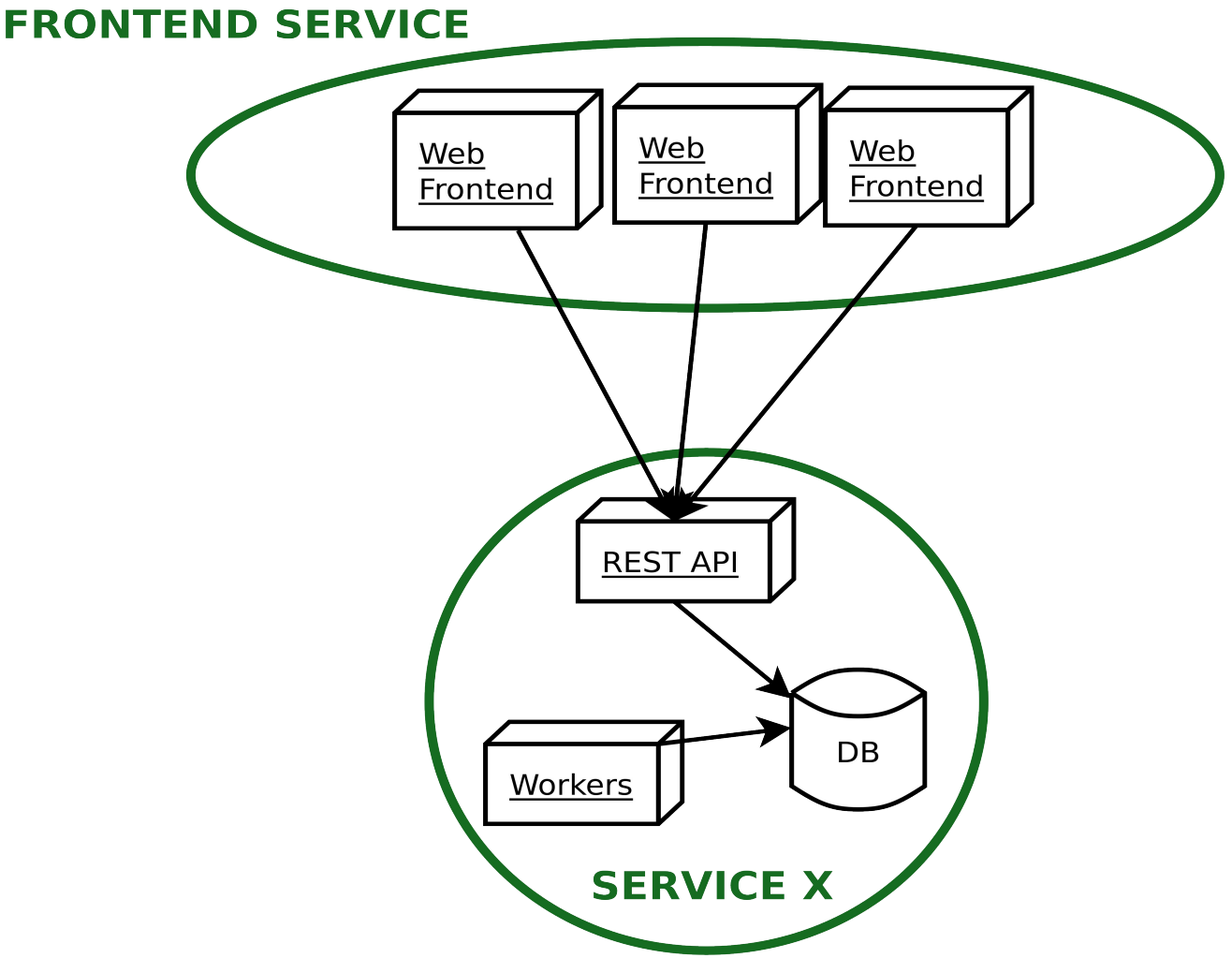
Intra-service policy

```
ACCEPT API to DB on TCP/5432

ACCEPT Workers to DB on TCP/5432
```



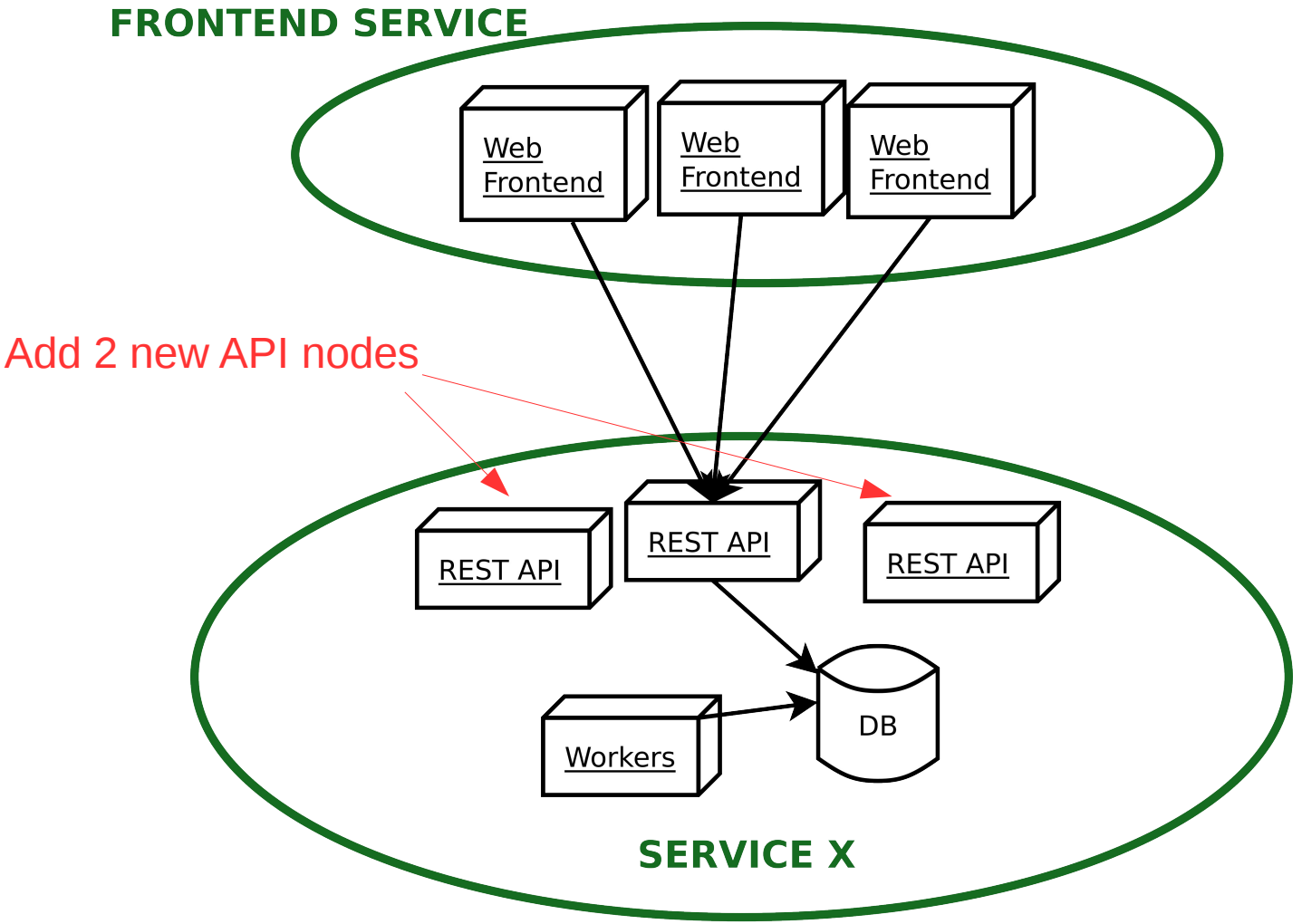
AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

Scalability (you know, the cloud)



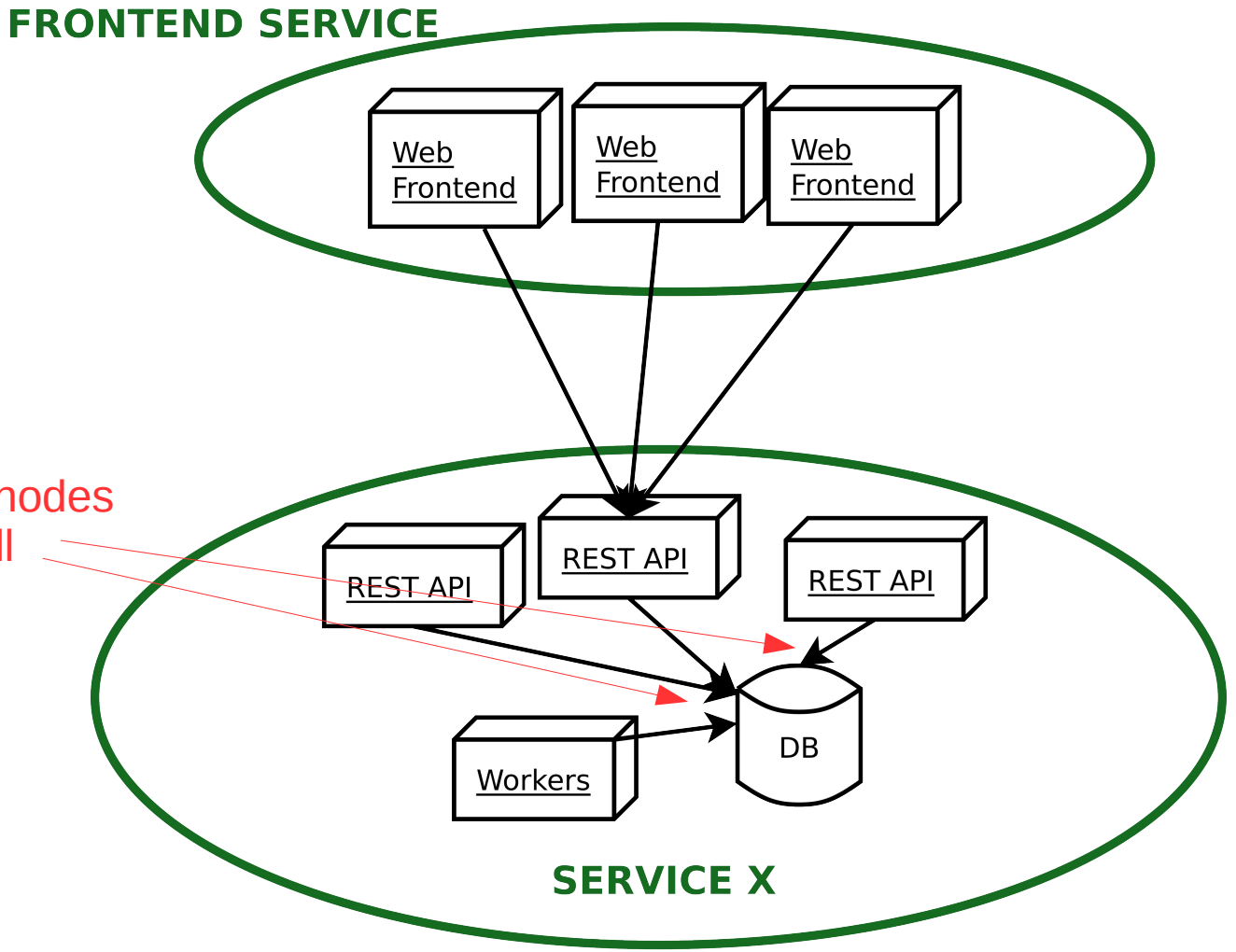
AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

Scalability (you know, the cloud)



AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

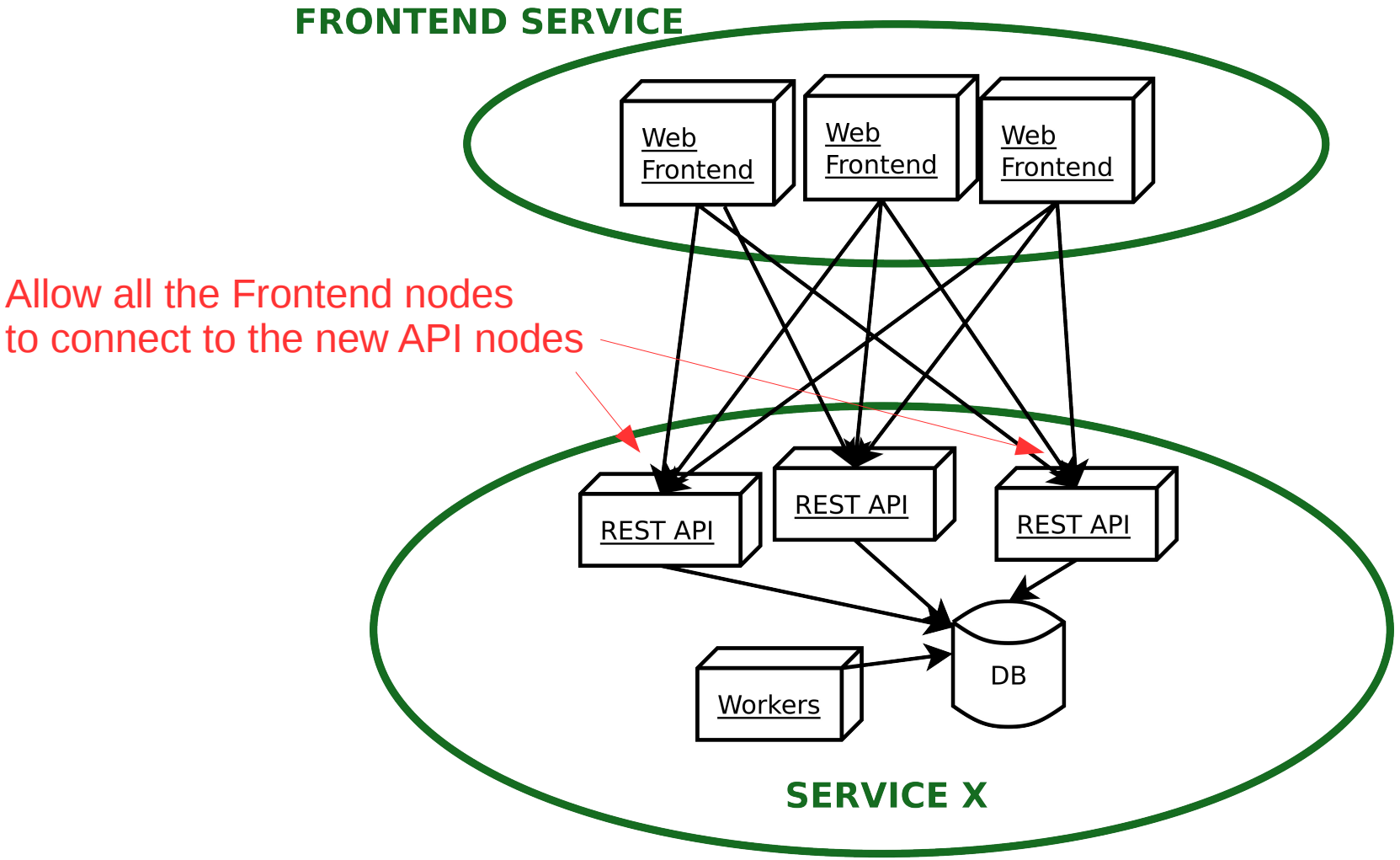
Scalability (you know, the cloud)



Allow the 2 API nodes in the DB firewall

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

Scalability (you know, the cloud)



The Tool: Chef

- Chef is a **provisioning** tool (puppet, cfengine, ol'-school-bash-script)
- **Cookbook**: ruby/chef scripts that installs and configure something.
- **Role**: a set of configuration value and a list of cookbooks to run.
- Administrator assigns a role to a **node** (server). Chef will run on the node, pull the list of cookbook and configuration variables, and install stuff on the node... and repeat again every X minutes.
- Files managed by Chef can't be edited manually.
- Each node indexes tons of metrics from running systems and store them in a central database (couchdb in Chef10, postgres in Chef11). It contains everything that you've ever dreamed of, and more !

Chef Searches


```
$ knife search node "roles:web-frontend AND chef_environment:staging"

3 items found
  Node Name:   frontend1.staging.domain
  Environment: staging
  FQDN:        frontend1.staging.domain
  IP:          172.21.1.2
  Run List:    role[base], role[web-frontend]
  Roles:       rsyslog-client, snmp-base, nagios-client, ntp-client,
  chef-client, ossec-agent, openldap-client,   web-frontend
  Recipes:     ohai, timezone, ntp, afw, apt,  system-tools, sysctl,
  nagios::client, snmp, diamond,   openldap::client, sudo, rsyslog,
  ossec::agent, nginx, varnish
  Platform:    centos 6

  Node Name:   frontend2.staging.domain
  Environment: staging
  FQDN:        frontend1.staging.domain
  IP:          172.21.1.3
  Run List:    role[base], role[web-frontend]
...
```

Chef Searches

```
# Get all the agents at once, more efficient
ossec_agents = search(:node,
                      "roles:ossec-agent AND chef_environment:prod")

ossec_agents.each do |agent|
  # this agent is running fine, go to the next one
  if ossec_agent_is_active?(agent_hash[:id])
    node.set[:ossec][:agents][agent_hash[:id]][:status] = "active"
    next
  else
    create_ossec_agent(agent_hash[:id])
    # Etc...
  end
end
```

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

Because Chef can search the entire infrastructure, it can be used to generate a firewall policy dynamically.

All we need is a syntax to declare the policy, and a cookbook to apply it.

Meet **AFW**

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

A{daptative,utomated,Weber,...} FireWall

- **Concepts**
  - Automated ruleset generation
  - 1 to 1 rules only: connection from one node to another is represented by one rule (no range opening)
  - User-specific outbound firewall: one user, identified by **UID**, can connect to one ip:port destination
  - Generic rules: avoid writing custom rules for each node, write rules for type of service instead
- **Technology**
  - Stock iptables-save format
  - Reload the ruleset every time chef runs, flushes unwanted rules
  - Netfilter features:
    - Fast reload: iptables-restore
    - Owner match (xt_owner)
    - Conntrack (xt_conntrack)

AFW: the syntax

- Rules are ***attributes*** of the AFW cookbook, and can be defined in roles, or other cookbooks

- Open INPUT access to rabbitmq

  port to a small list of servers

```
'RabbitMQ AMQP Producers' => {
  :direction => 'in',
  :user => 'rabbitmq',
  :protocol => 'tcp',
  :interface => 'default',
  :source => ['producer1.production.domain',
              'producer2.production.domain',
              'producer1.staging.domain'],
  :dport => '5672'
},
```

- Same, but in the staging

  environment only

```
'MongoDB Staging access from Jenkins' => {
  :direction => 'in',
  :protocol => 'tcp',
  :user => 'mongodb',
  :dport => '27017',
  :source => ['jenkins1.production.domain',
              'jenkins2.production.domain'],
  :interface => 'all',
  :env => 'staging'
}
```

AFW: Searches

- Uses Chef's search capabilities to list the nodes allowed to connect.
- Open the firewall between a server and its clients. On the right, for Ossec.
- Open a backend database to application servers, below for mongodb.

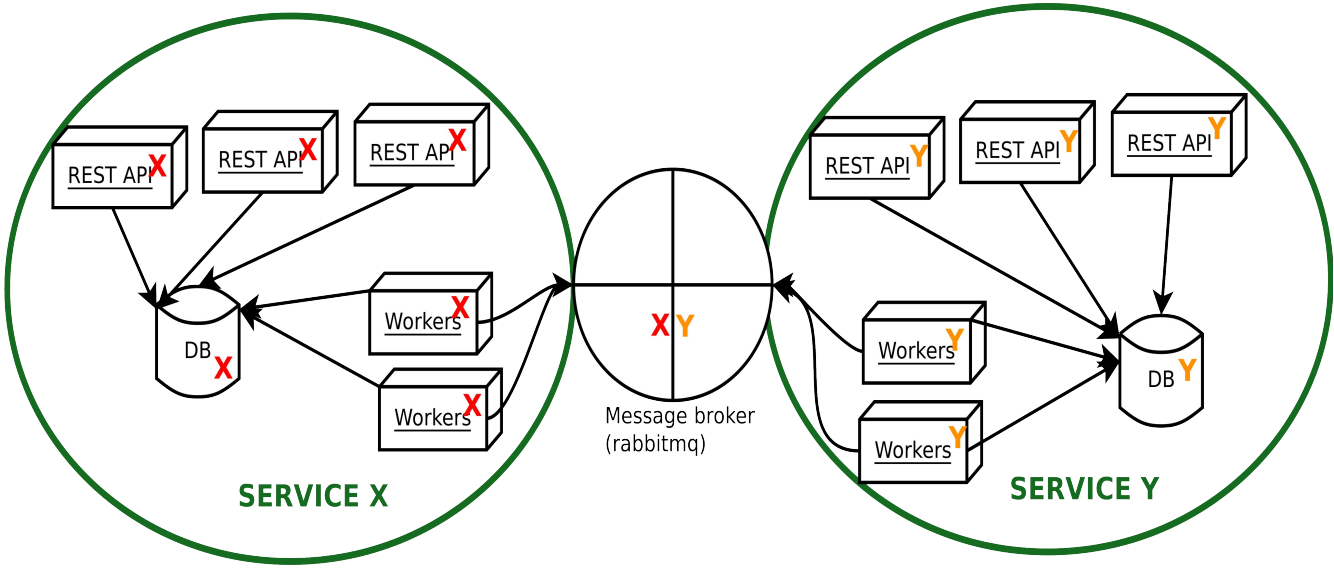
```
'MongoDB App Entry Point' => {
  :protocol => 'tcp',
  :direction => 'in',
  :user => 'mongodb',
  :source => '(roles:nodejs
              OR roles:python-worker
              OR roles:api-server)'
  :dport => '27017'
},
```

```
default_attributes(
  :afw => {
    :rules => {
      'ossec_agent_to_server' => {
        :direction => 'in',
        :protocol => 'udp',
        :user => 'ossec',
        :dport => '1514',
        :source => 'roles:ossec-agent'
      },
      'ossec_server_to_agent' => {
        :direction => 'out',
        :protocol => 'udp',
        :user => 'ossec',
        :dport => '1514',
        :destination => 'roles:ossec-agent'
      }
    }
  },
```

AFW Firewalling dynamic infrastructures (the cloud) with Chef and Netfilter

AFW: The notion of service

- AFW rules are generic across all services
- **tag** identifies the members of a service
- Ex: database accepts connections from the app servers in its service



All nodes in Service X are tagged `X`.
All nodes in Service Y are tagged `Y`.
The message broker has both `X` and `Y` tags

```
'Accept connections from app servers' => {
  :protocol => 'tcp',
  :direction => 'in',
  :user => 'postgres',
  :destination => 'roles:app-server AND SAMETAG',
  :dport => '5432'
},
```

```
roles:mongodb AND tags:X AND chef_environment:#{node.chef_environment}
```

```
→ returns: ['REST-API-X1','REST-API-X2','REST-API-X3',
            'Worker-X1', 'Worker-X2']
```

AFW: firewall a mongodb cluster

- When building a MongoDB cluster, all members share the same “shard”.
- Chef knows the name of the shard, in the **shard_name** attribute.
- **shard_name** is used in a source/destination search to find the members of a cluster, and open the firewall to them

```
'MongoDB Cluster Inbound Replication' => {
  :protocol => 'tcp',
  :direction => 'in',
  :user => 'mongodb',
  :source => 'shard_name:#{node[:mongodb][:shard_name]}',
  :dport => '27017'
},
'MongoDB Cluster Outbound Replication' => {
  :protocol => 'tcp',
  :direction => 'out',
  :user => 'mongodb',
  :destination => 'shard_name:#{node[:mongodb][:shard_name]}',
  :dport => '27017'
},
```

AFW: Unusual rules ? Predefine them.

- Predefined rules are copied verbatim into the iptables ruleset. No interpretation.

```
:afw => {
  :rules => {
    'Accept all packets router through the bridge' => {
      :table => 'filter',
      :rule => '-I FORWARD -o br0 -m physdev --physdev-is-bridged -j ACCEPT'
    },
    'Drop connection to the admin panel on the eth0 interface' => {
      :table => 'mangle',
      :rule => '-A INPUT -i eth0 -p tcp --dport 80 -m string --string "get /admin
http/1.1" --icase --algo bm -m conntrack --ctstate ESTABLISHED -j DROP
    },
    'DNAT a source IP to change the destination port' => {
      :table => 'nat',
      :rule => '-A PREROUTING -i eth3 -s 201.23.72.3 -p tcp --dport 8008 -j DNAT
--to-destination 127.0.0.1:1234'
    },
    'Dont do conntrack on this specific user's UDP packets' => {
      :table => 'raw',
      :rule => '-A OUTPUT -o eth0 -p udp -m owner --uid-owner 105 -j NOTRACK'
    }
  }
}
```

AFW: Limitations

- The entire security relies on the security of the Chef server (true for all provisioning systems)

- Nodes can modify their own Chef attributes. If **one node gets hacked**, it can modify its run_list, environment and tags to impersonate another node !

```
root@hackedserver# shef -z
  chef > node.tags
   => ["foo"]
  chef > node.tags.push("bar")
   => ["foo", "bar"]
  chef > node.save
   => <Chef::Node:0x3fc6a2c3b830 @name="hackedserver.domain.net">
  chef > node.tags
   => ["foo", “bar”]
```

Future Work

- Ipset
  - If a search returns more than {10? 20 ? 100?} IPs, automatically create an Ipset.
- Forward rules
  - Use AFW to manage the FORWARD rules of a border firewall
- IPv6
- Ebtables
- Support for more modules (time, string, …)

Future Work: Service Oriented Security

- Instead of managing the firewall as a network policy, manage it as an Access Control List

```
"application": {
  "accounting": {
    "dependencies": {
      "applications": [ "printing", "human-ressources" ],
      "infrastructure": [ "graphite", "internal-smtp" ]
      "external": [ "https://api.salesforce.com",
                    "https://api.paypal.com" ]
    }
  },
  "human-ressources": {
    "dependencies": {
      …
    }
```

Questions ?

- **https://github.com/jvehent/AFW/**
- **http://community.opscode.com/cookbooks/afw**

AFW: Open rules “on-the-fly”

- When we can't create generic rules ahead of time, we can let a cookbook create its own rules.
- AFW's core functions can be called from another cookbook. It's all Ruby.

```
# Call the AFW module to create the rule
AFW.create_rule(node,
                "Haproxy local LB to #{server['hostname']}:#{port}",
                {'protocol' => 'tcp',
                 'direction' => 'out',
                 'user' => 'haproxy',
                 'destination' => "#{server['ipaddress']}",
                 'dport' => "#{port}"
                })
```

- Useful, but harder to diagnose. Use with caution.

AFW: User specific Outbound rules

- Netfilter's INPUT chain cannot check the owner of the socket
- But the OUTPUT chain can:
  - Example with the `root` user

```
-A OUTPUT -m owner --uid-owner 0 -m state --state NEW -j root
:root – [0:0]
# Root user is allowed to connect to the RSYSLOG server
-A root -o eth1 -p tcp --dport 514 -d 172.31.15.13 -m conntrack --ctstate NEW -j ACCEPT
# Root is also allowed to connect anywhere in TCP
-A root -p tcp --dport 1:65535 -d 0.0.0.0/0 -m conntrack --ctstate NEW -j ACCEPT
# Everything else is logged
-A root -j LOG --log-prefix "DROP_AFW_OUTPUT_root " --log-uid --log-tcp-sequence
```

  - The `nagios` user has much less permissions

```
-A OUTPUT -m owner --uid-owner 107 -m state --state NEW -j nagios
:nagios – [0:0]
# Nagios local user is allowed to connect to the Nagios server
-A INPUT -i eth1 -p tcp --dport 5666 -s 172.31.12.42 -m conntrack --ctstate NEW -j ACCEPT
-A nagios -j LOG --log-prefix "DROP_AFW_OUTPUT_nagios " --log-uid --log-tcp-sequence
```