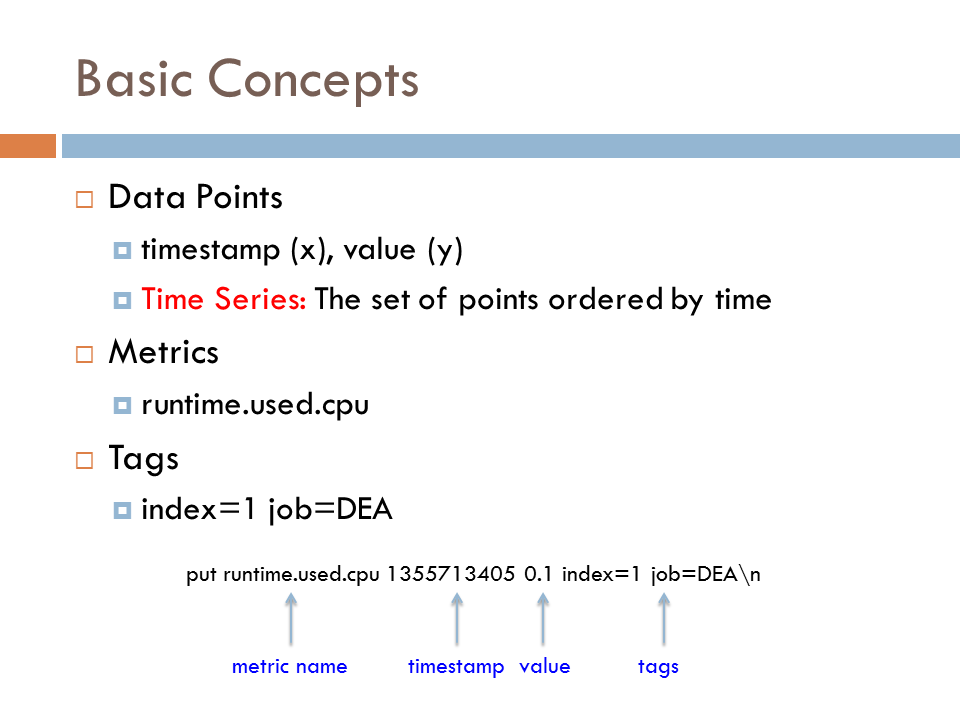
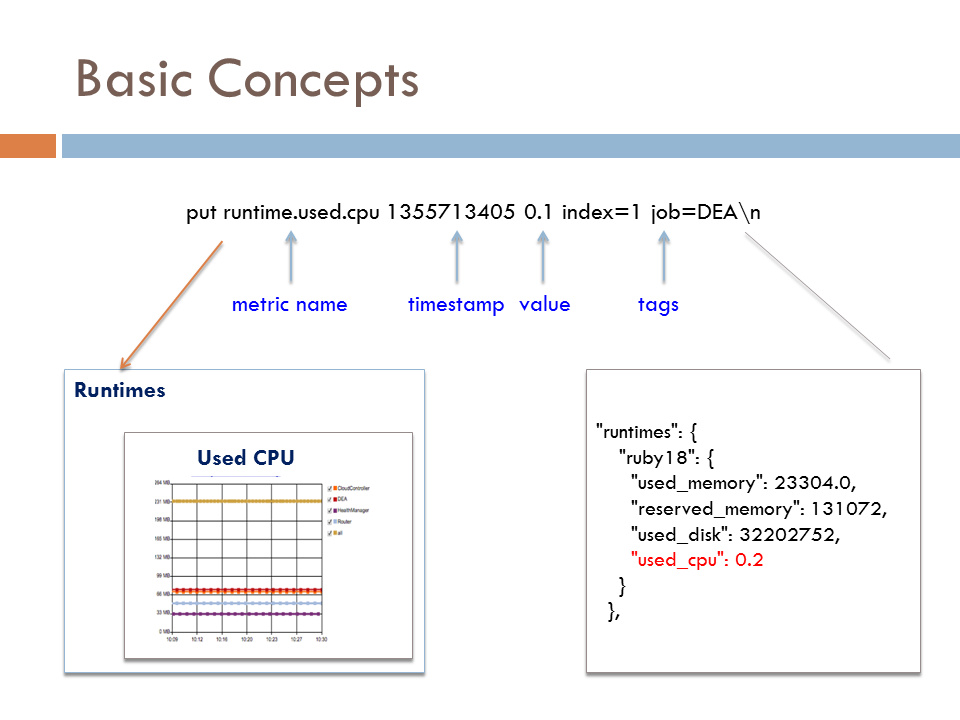
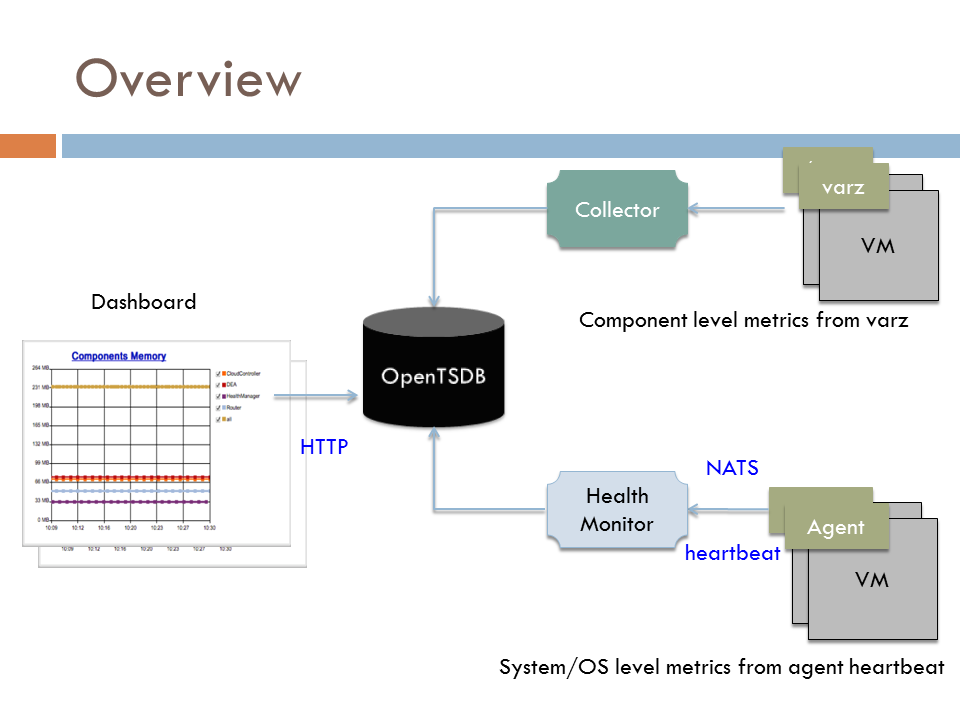
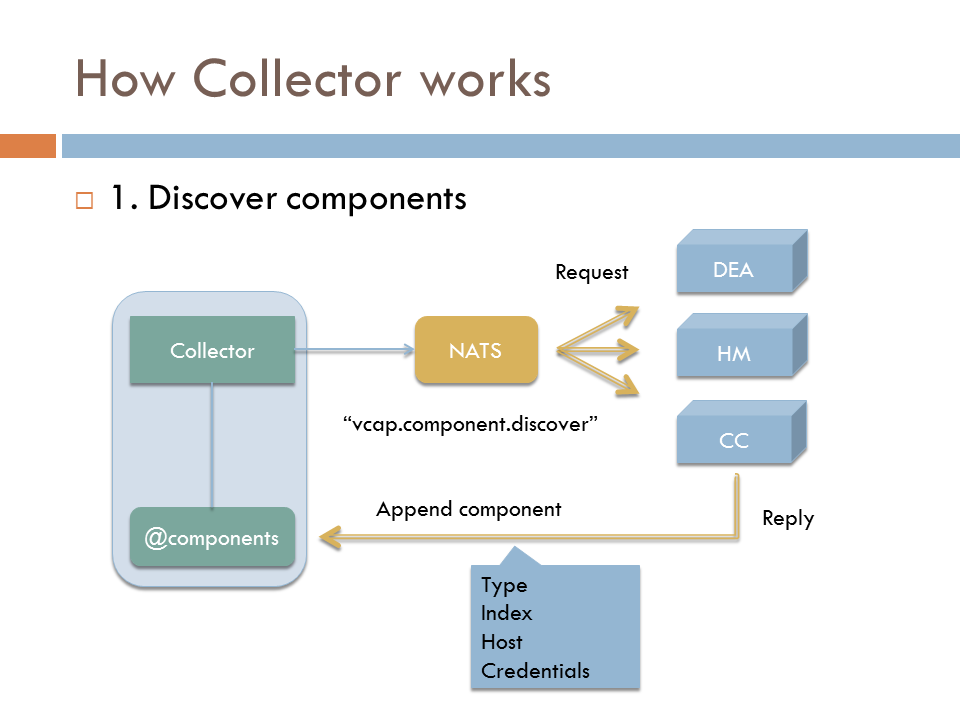
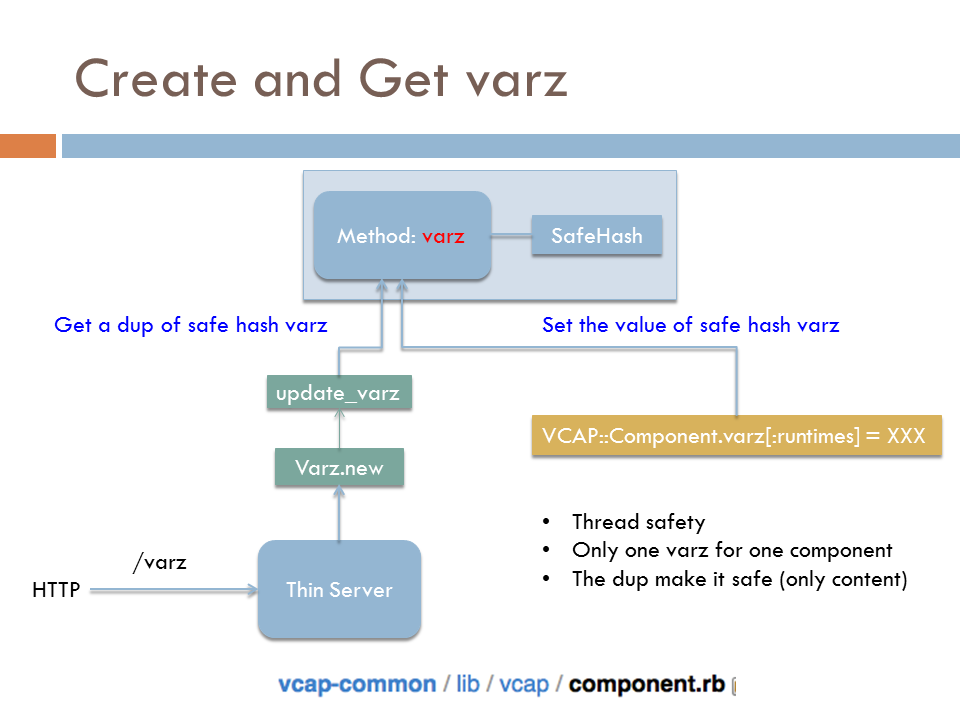
# [Monitoring Cloud Foundry On System Level](http://blog.csdn.net/resouer/article/details/8709609)

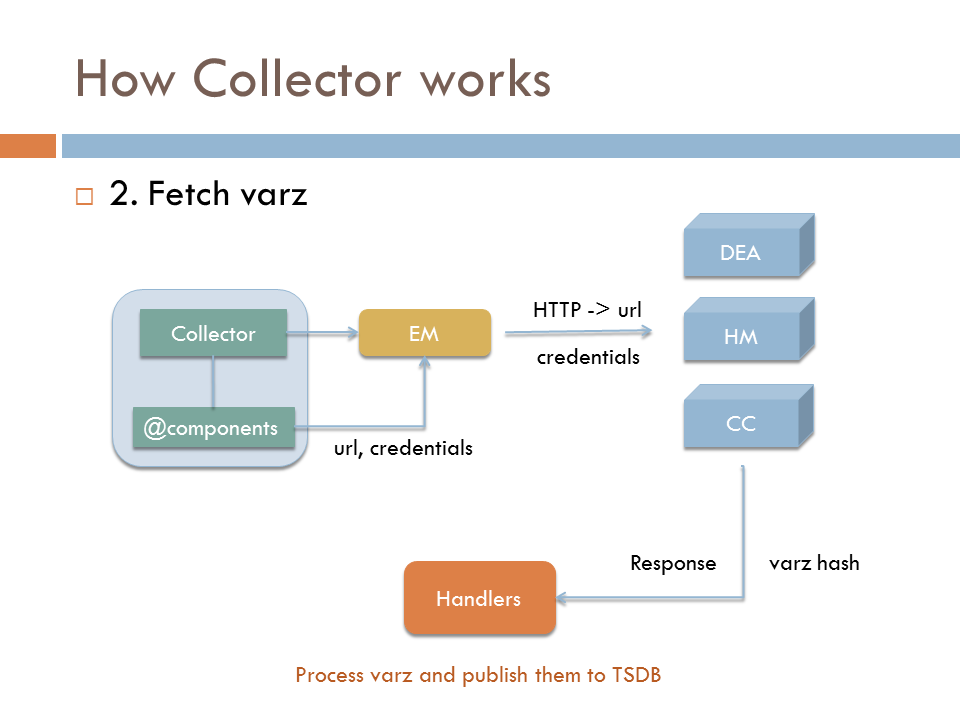
<http://blog.csdn.net/resouer/article/details/8709609>

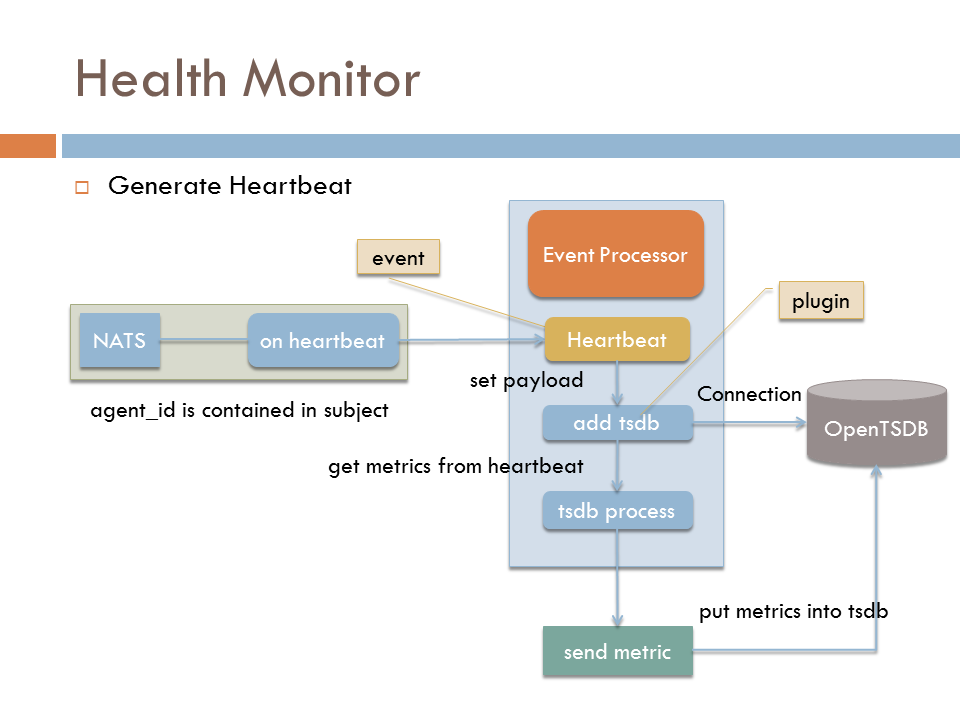
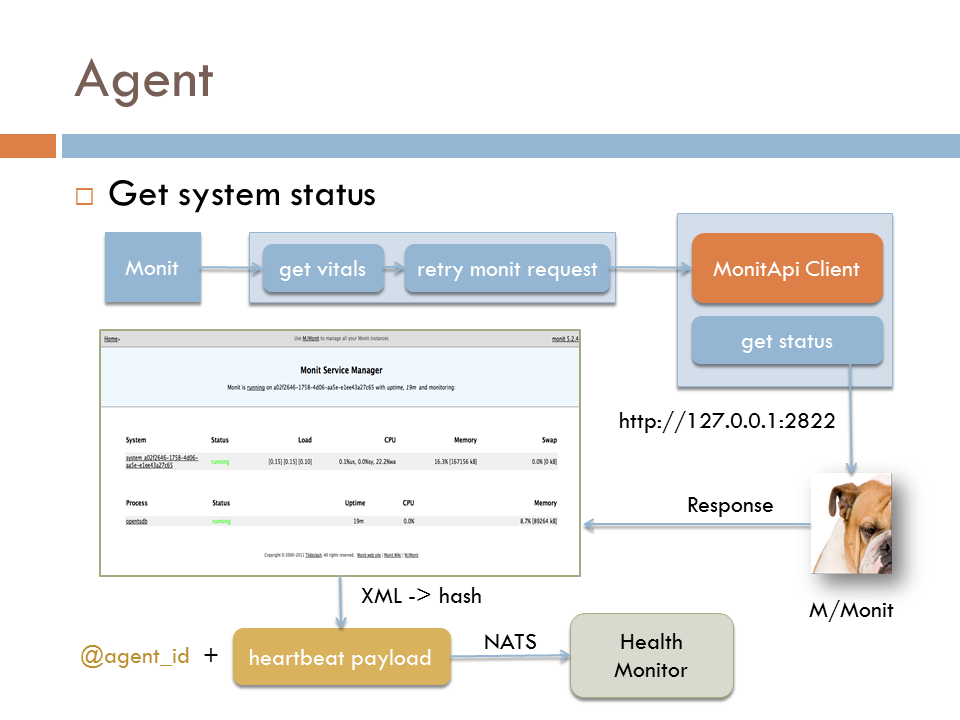
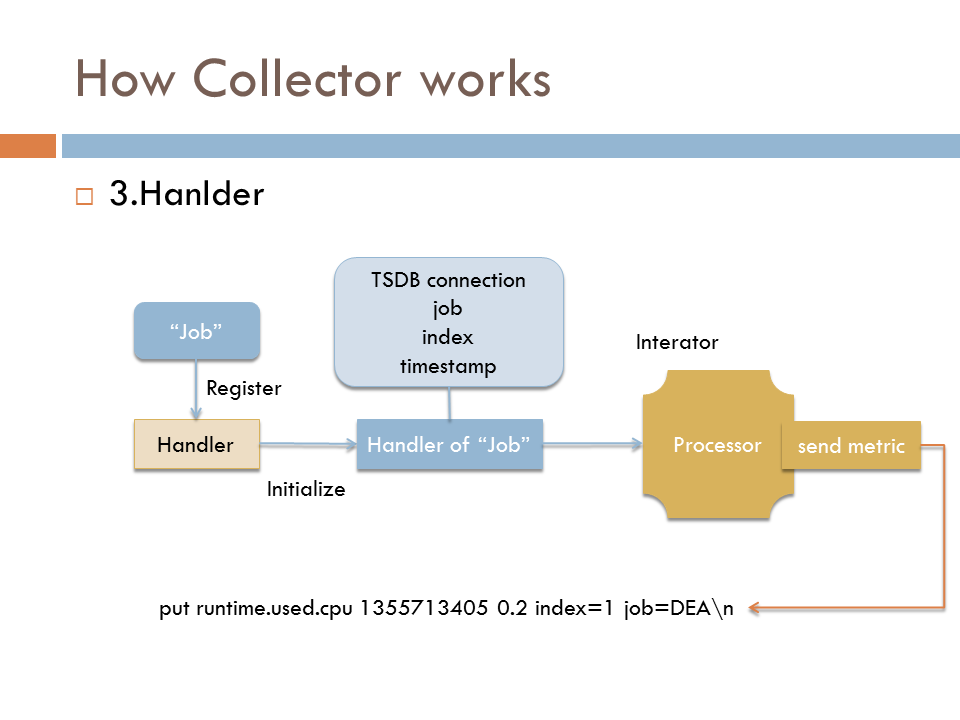
**NOTE: This is a summary of my previous work on Cloud Foundry monitoring system based on BOSH.**

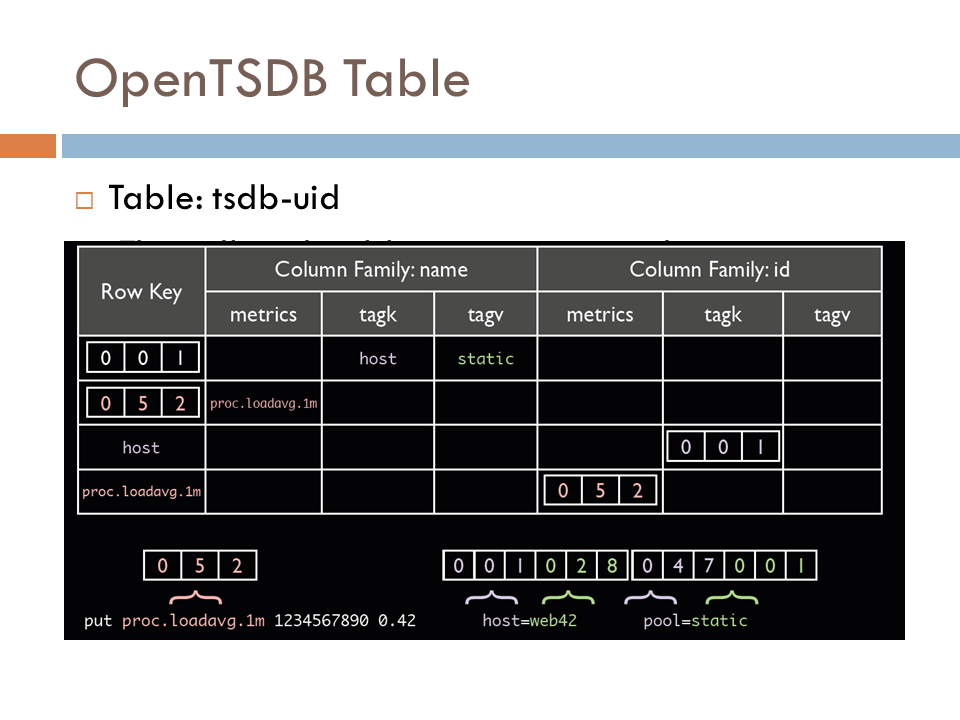
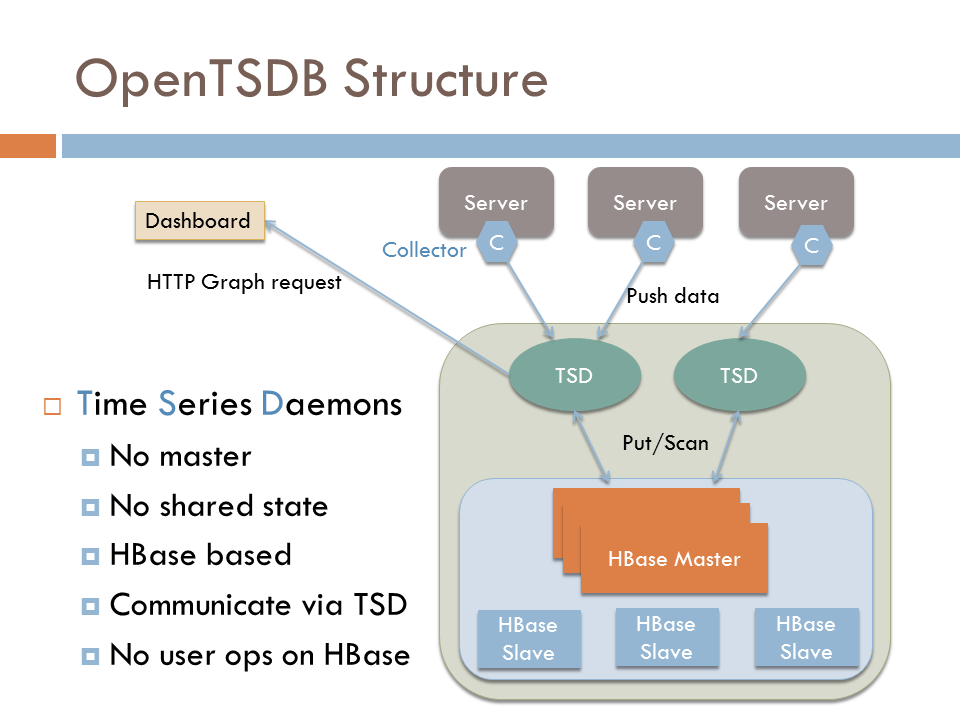
There are two levels for monitoring CF based on BOSH: one for varz and one for agent; one returns stats of components and the other, the stats of underlying system. We'll see how the to collect varz from all the components and how Health Monitor get the system info from monit (M/Monit) part of BOSH agent.

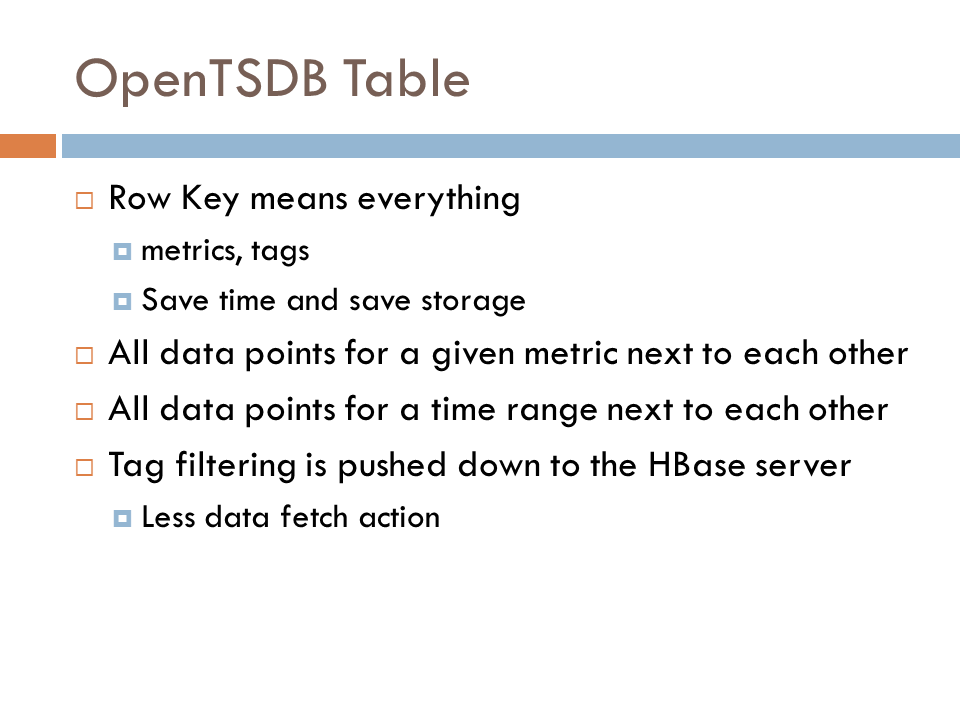
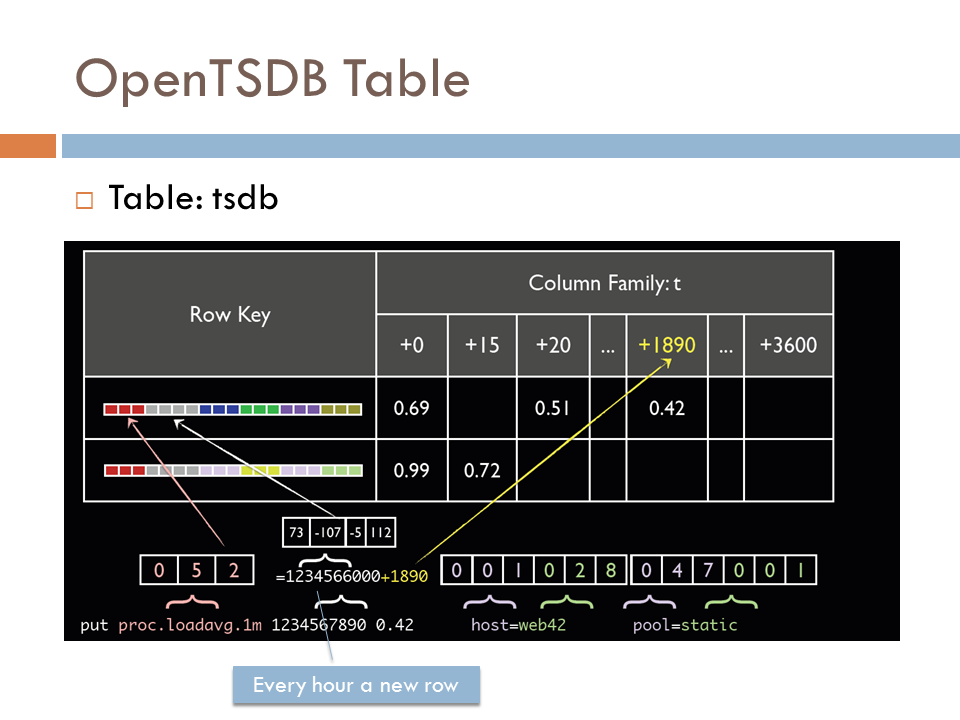












# [How to collect stats of apps from CloudFoundry and do analysis?](http://blog.csdn.net/resouer/article/details/7866753)

Suppose we have apps running on CloudFoundry, how can we get the stats of app and doing some analysis here?

vmc stats appname is really useful. But what we need is a component which returns a json result according to a sepecific URL.

According to the code of CF, we can find there's something useful here:

<https://github.com/cloudfoundry/vcap-common/blob/master/lib/vcap/component.rb#L44>

#Common component setup for discovery and monitoring

During the start process of router, an endpoint of var component will be startup for /varz or /healthz

The method register(opts) will be called in router.rb:

1. # Register ourselves with the system
2. status\_config = config['status'] || {}
3. VCAP::Component.register(:type => 'Router',
4. :host => VCAP.local\_ip(config['local\_route']),
5. :index => config['index'],
6. :config => config,
7. :port => status\_config['port'],
8. :user => status\_config['user'],
9. :password => status\_config['password'],
10. :logger => Router.log)

The method to start a  server is below, so if we can request this server with /varz, the stats will be returned.

1. **def** start\_http\_server(host, port, auth, logger)
2. http\_server = Thin::Server.**new**(host, port, :signals => **false**) **do**
3. Thin::Logging.silent = **true**
4. use Rack::Auth::Basic **do** |username, password|
5. [username, password] == auth
6. **end**
7. map '/healthz' **do**
8. run Healthz.**new**(logger)
9. **end**
10. map '/varz' **do**
11. run Varz.**new**(logger)
12. **end**
13. **end**
14. http\_server.start!
15. **end**

We can see Thin is is used here to start a http server.

So, we need to figure out the port and host on which the server is running. Actually, this things is customizable. We can find this in dea.yml

1. # Used for /healthz and /vars endpoints. If not provided random
2. # values will be generated on component start. Uncomment to use
3. # static values.
4. #status:
5. #  port: 34501
6. #  user: thin
7. #  password: thin

That means we can specify its port and auth before we start a DEA, just add status part in your config file, and then:

**Sending http request to that server will return you a JSON response which contains a key named "running apps", its value what you need！**

What's more, in

.../cloudfoundry/.deployments/devbox/deploy/rubies/ruby-1.9.2-p180/lib/ruby/gems/1.9.1/gems/vcap\_common-1.0.10/lib/vcap/component.rb

we can see how these parameters before to be used to register itself into your local vcap environment:

1. **def** register(opts)
2. uuid = VCAP.secure\_uuid
3. type = opts[:type]
4. index = opts[:index]
5. uuid = "#{index}-#{uuid}" **if** index
6. host = opts[:host] || VCAP.local\_ip
7. port = opts[:port] || VCAP.grab\_ephemeral\_port
8. nats = opts[:nats] || NATS
9. auth = [opts[:user] || VCAP.secure\_uuid, opts[:password] || VCAP.secure\_uuid]
10. logger = opts[:logger] || Logger.**new**(**nil**)
12. # Discover message limited
13. **@discover** = {
14. :type => type,
15. :index => index,
16. :uuid => uuid,
17. :host => "#{host}:#{port}",
18. :credentials => auth,
19. :start => Time.now
20. }
22. # Varz is customizable
23. **@varz** = **@discover**.dup
24. **@varz**[:num\_cores] = VCAP.num\_cores
25. **@varz**[:config] = sanitize\_config(opts[:config]) **if** opts[:config]
27. **@healthz** = "ok\n".freeze
28. ... ...

But the rest of this part requires some knowledge of EventMachine. So we'll talk about it later.