モンストを支えるインフラの今とこれから

2016.3.1 dots. Conference Spring 2016

ゲーム開発の裏側

mixi, Inc. XFLAG™ STUDIO

@isaoshimizu

About Me

- 清水 勲 SHIMIZU ISAO @isaoshimizu
- 2011年8月より株式会社ミクシィ
- 現在はエックスフラッグスタジオ システム開発部 所属
- 約2年前にソーシャル・ネットワーキング サービス mixi の運用から
  スマートフォンゲーム「モンスターストライク」の運用へ

モンスターストライク

- 2013年10月10日 正式リリース
- iOS版, Android版を提供
- 全世界で3,000万人以上がプレイ（同一端末で重複ダウンロードされた数は含まず）
- 日本、台湾、韓国、北米、香港・マカオで展開
- YouTubeでアニメの配信 2,500万回の再生、ニンテンドー3DS版 出荷数100万本突破

ここからはモンスターストライク
日本版に限定した内容になります

数値で見るモンスターストライクのインフラ

Application

300+ Servers

Redis

30 Servers

CDN Traffic

Max 270Gbps

Resque

50 Servers

Memcached

200+ Servers

API Traffic

Avg 1Gbps

TURN

40 Servers

MariaDB

250+ Servers

Internal Traffic

Max 20Gbps

インフラ構成

Main

Main/Backup

Main/Dev

複数DCとAmazon VPC

- DC 1（メイン）、DC 2（メインとバックアップ）、AWS（メインと開発環境）で構成
- 各DCはTY2,4経由でAmazon VPCとDirectConnectでプライベート接続
- 各DC間は片系障害時においても20Gbpsまで耐えられる設計
- 詳しい解説
  “拠点冗長した話（内部トラフィックボリューム編）”
  http://xflag.com/blog/infradb/internal_traffic_volume.html

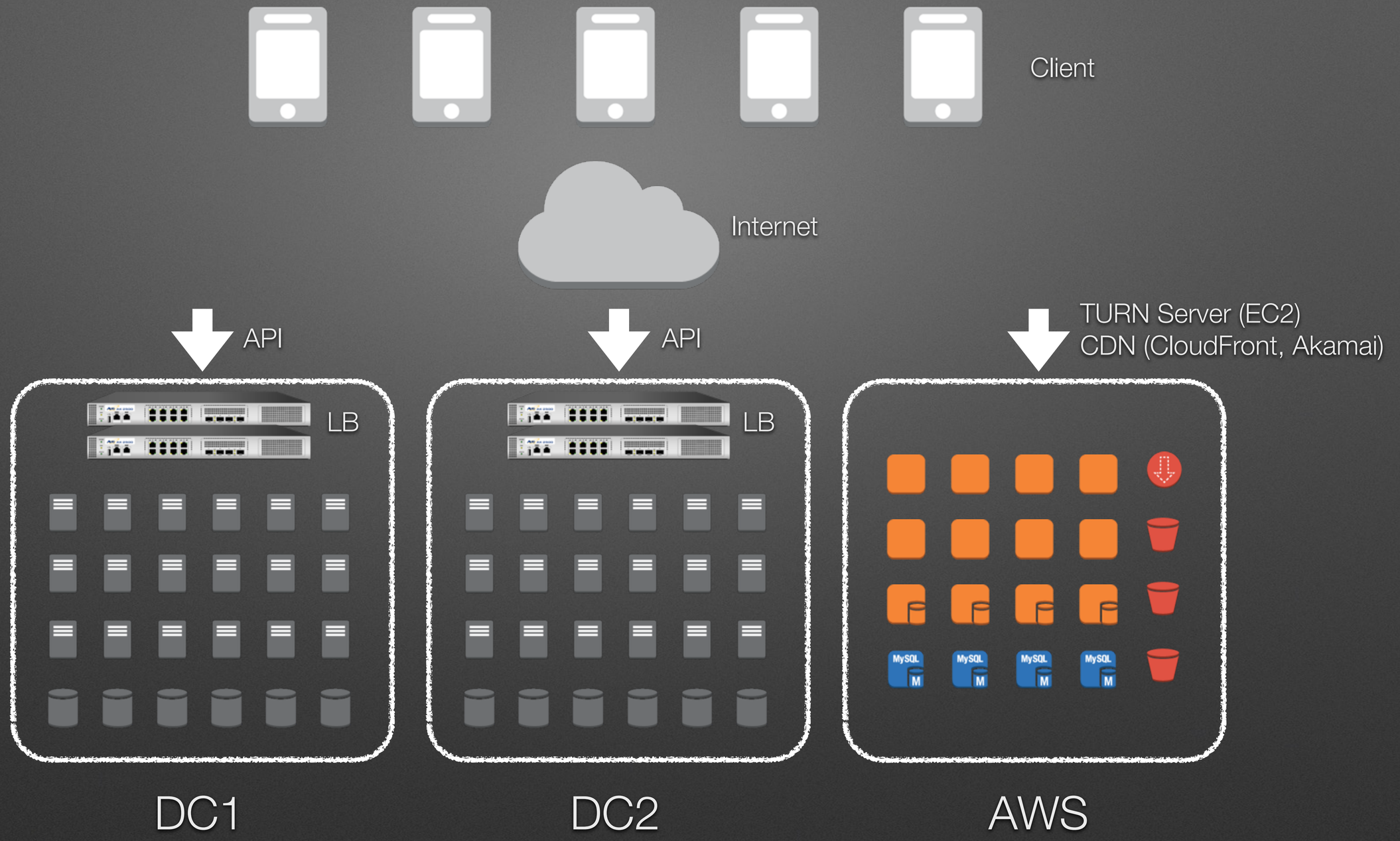
各拠点の役割

Main　　Main/Backup　　Main/Dev

各拠点の役割

- 本番のApplication、ResqueはDC1、DC2の両方で稼働
- 本番のRedis、MariaDB、Memcached
  - DC1にMaster/Backupのセット
  - DC2にBackup
  - BackupはDC1とDC2合わせて2セットある
- AWSでは本番のTURN Server（マルチプレイ用）、開発用インスタンス群などが稼働。

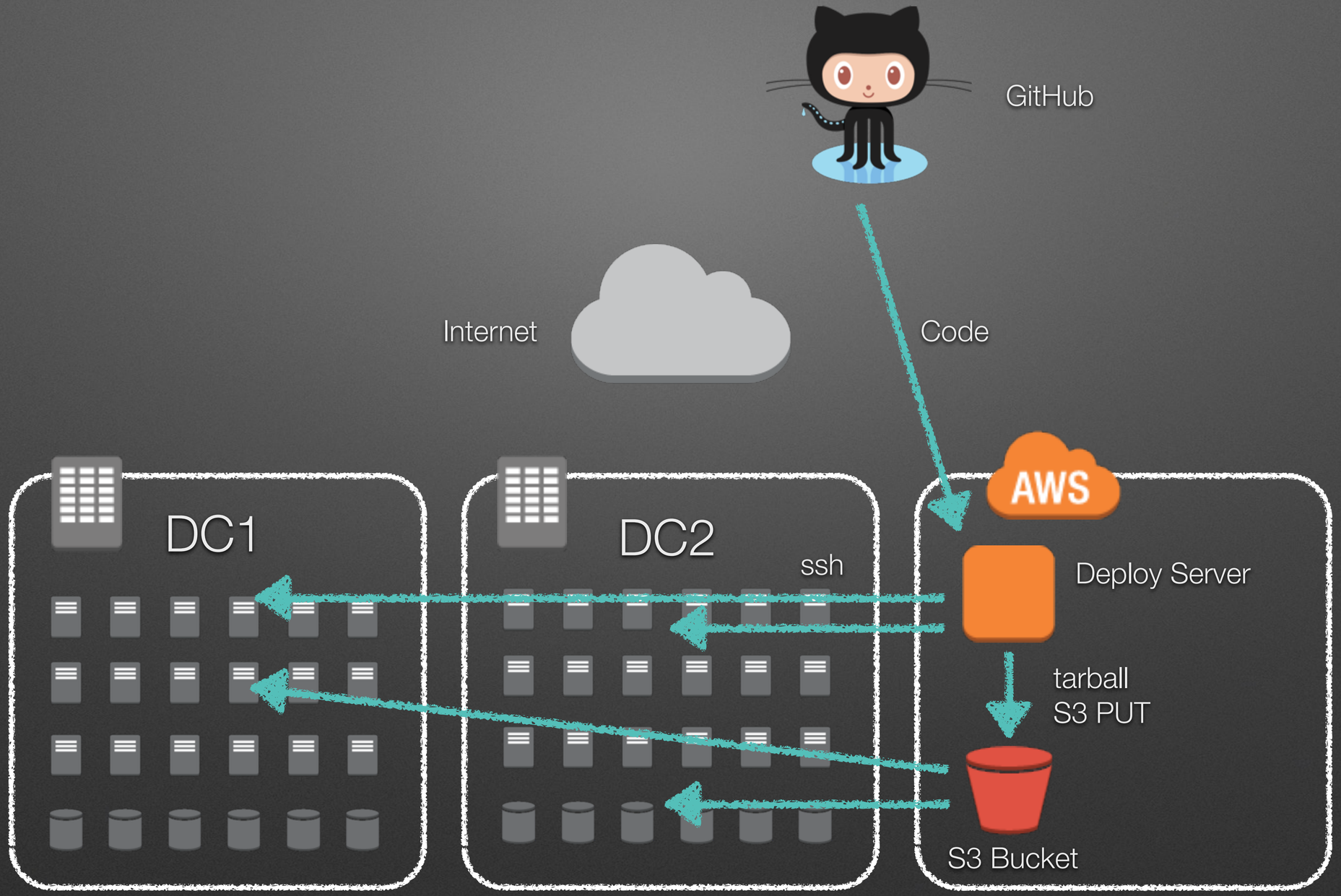
クライアントアプリから見たインフラ


Client
Internet
API
API
TURN Server (EC2)
CDN (CloudFront, Akamai)
LB
LB
MySQL
M
MySQL
M
MySQL
M
MySQL
M
DC1
DC2
AWS

クライアントアプリからのアクセス先

- DC1とDC2にあるロードバランサー（HA構成）へ
- ロードバランサーへのアクセスはDNSラウンドロビン（2つの仮想IP）
- ロードバランサーから各拠点のApplicationサーバへ振り分け
- AWSではマルチプレイ用にTURN Server（EC2 + Elastic IP）
- ゲームデータ・リソースデータ用にCDN（CloudFront + S3）。Akamaiも併用。

デプロイ

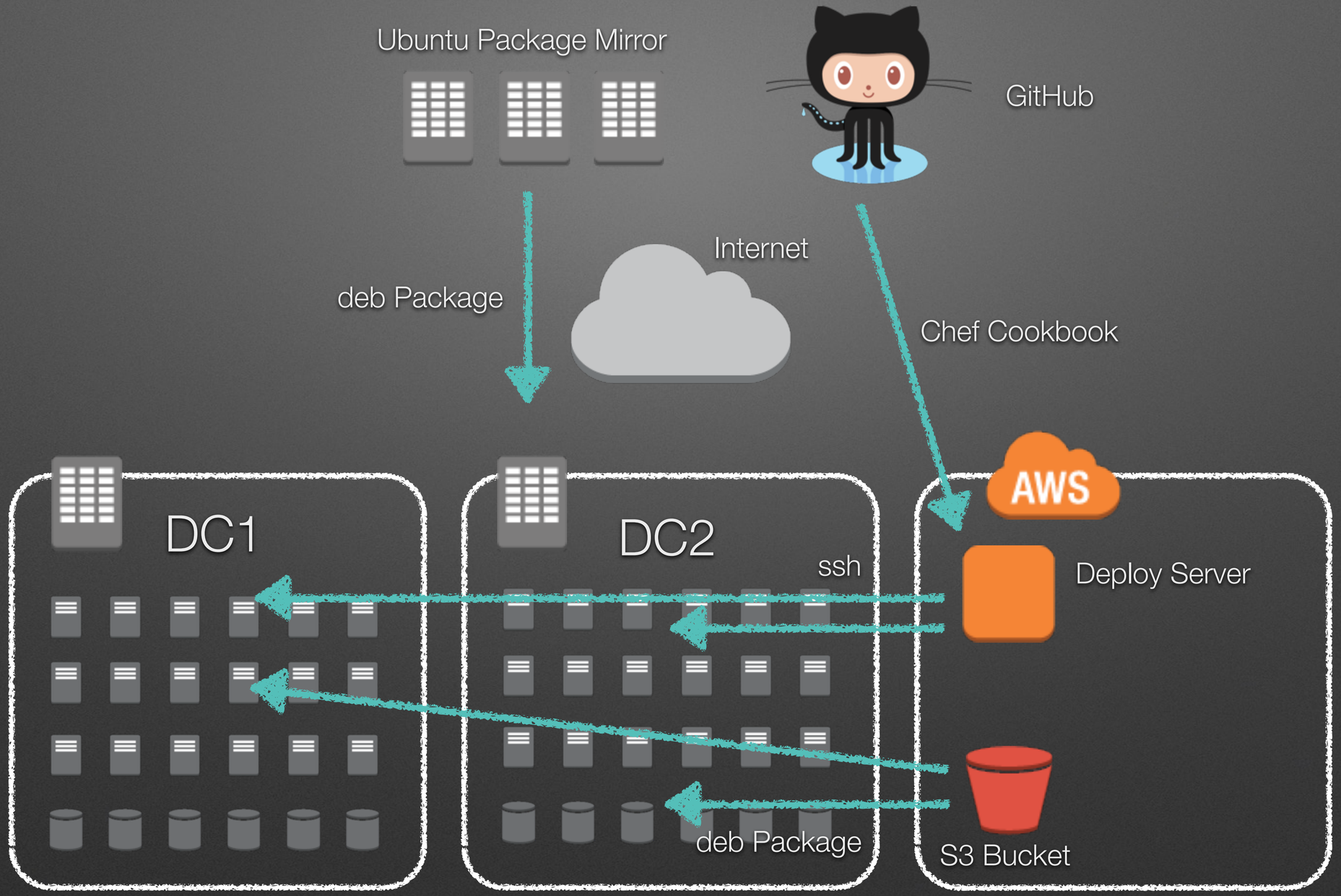

GitHub
Internet
Code
AWS
DC1
DC2
ssh
Deploy Server
tarball
S3 PUT
S3 Bucket

デプロイ

- Capistrano 2
- 必要なときに必要なだけデプロイする
- capistrano-s3copy-awscli https://github.com/bacter1a/capistrano-s3copy-awscli
  - GitHubから最新のコードを取得してS3アップロード用にtarballを作成
  - ApplicationサーバではS3にアップロードされたtarballをダウンロード、展開
  - GitHubへの接続数制限対策（github.comへ大量に接続が要求されると拒否されるため）

- capistrano-rbenv https://github.com/yyuu/capistrano-rbenv
  - バージョン指定でRubyの配布、アップデートが楽に
  - Ruby 2.2.4導入済み

プロビジョニング


Ubuntu Package Mirror
GitHub
Internet
deb Package
Chef Cookbook
AWS
DC1
DC2
ssh
Deploy Server
deb Package
S3 Bucket

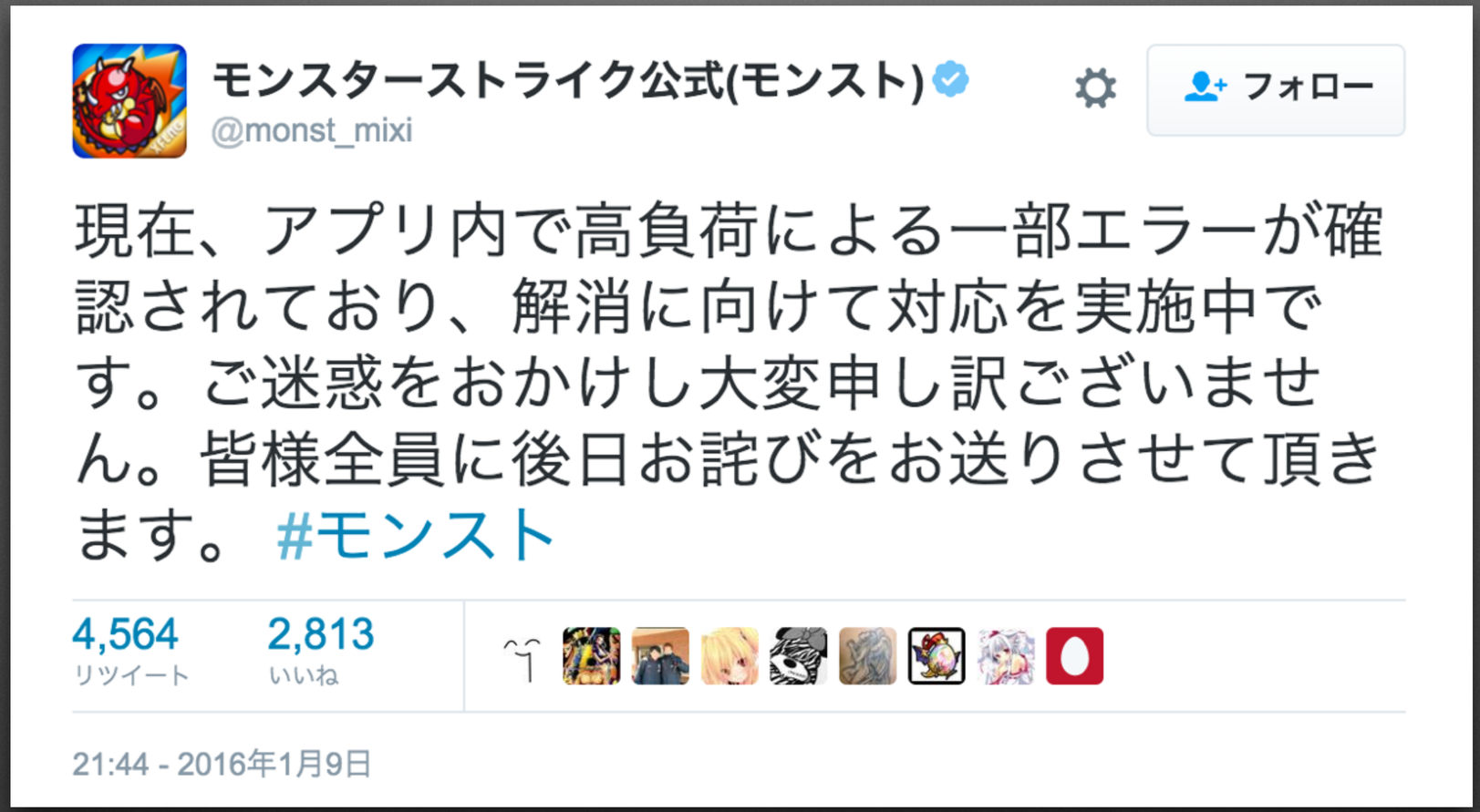
プロビジョニング

- Chefによるプロビジョニングが基本
  - capistrano-paratrooper-chef
    https://github.com/tk0miya/capistrano-paratrooper-chef
- 物理マシンの初期セットアップ（最低限）はAnsibleを使う
  - OSクリーンインストール直後の状態では、Chefがインストールされていないため

- Community Cookbookをincludeした独自Cookbookを作成、利用している
- 独自Cookbookは、XFLAG全体のインフラでも共通して使えるように
- すべてのサーバがChefで構築可能
- 稼働中の本番DB、Memcachedに再適用しても問題が起きないように

- S3にMariaDBのレポジトリのミラー（本家の障害対策）
  - ioDriveなどのドライバも置いてある
  - aptlyで構築 http://www.aptly.info/
- 個人の開発マシンでもVagrant + VirtualBox + Chefで構成できるように
- Test Kitchenの活用 https://github.com/test-kitchen/test-kitchen

負荷対策



今年に入ってからも高負荷が続く... 🤦

負荷対策

- DBのシャーディング、テーブル分割を継続実施
- Applicationサーバの増設
  12-20 core なCPU x 2（Hyper-Threading換算で24-40 core）のマシンが基本構成
- クエリ改善やキャッシュ対応
  ioDrive(ioMemory)からSSDへのスケールダウンの実現
- Memcachedの増設

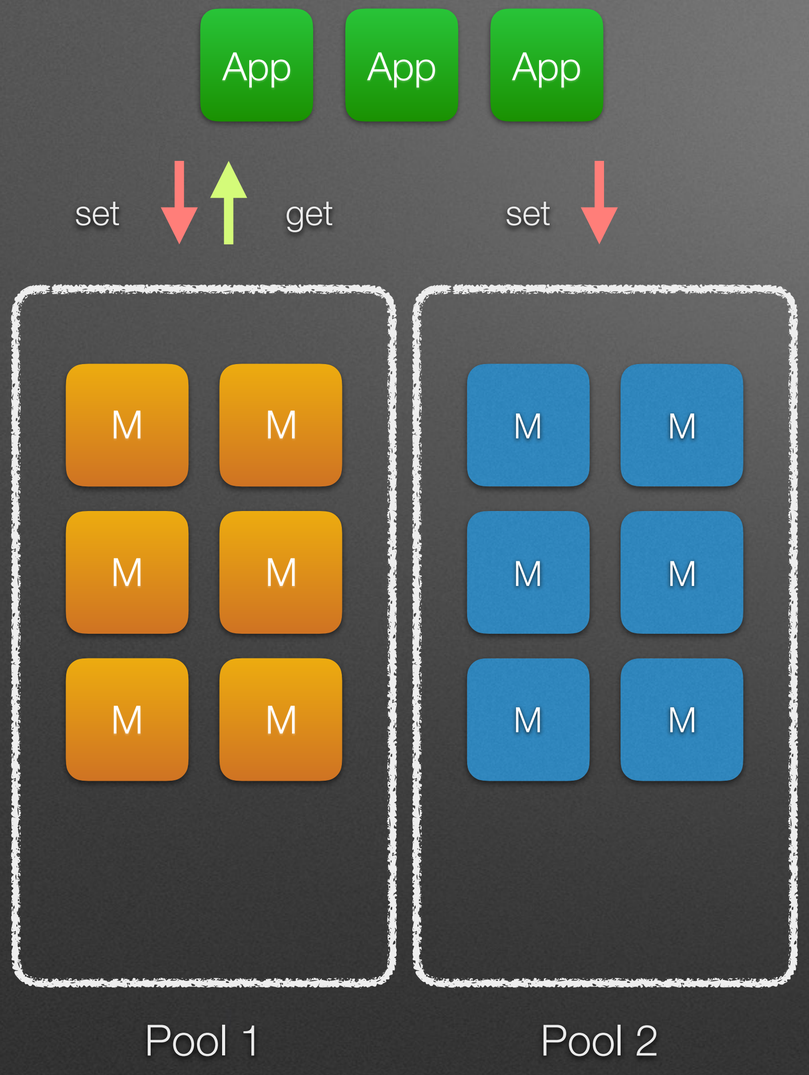
Memcachedの構成

Memcachedの構成

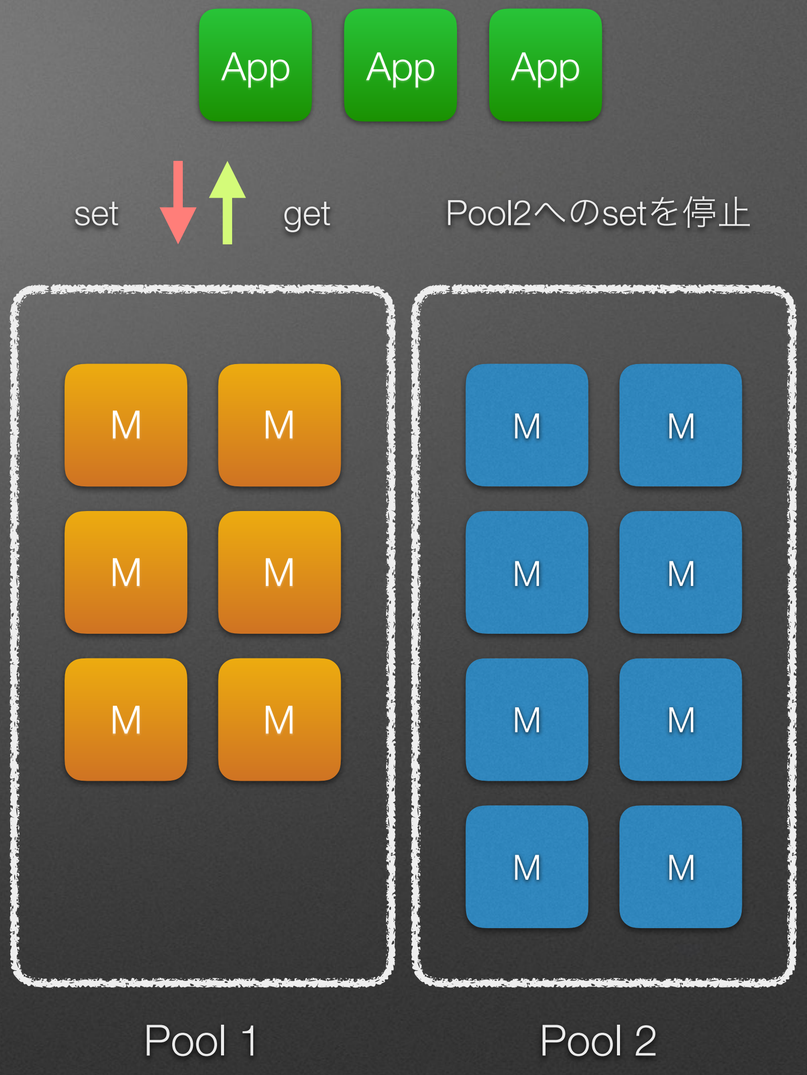
- Memcachedは2プール構成
- 1プールあたり約100台
- マシン1台あたりのメモリ割り当て26GB（メモリ32GB搭載のマシン）
  - 1プール全体で約2.6TBの容量をもつ
- 両方のプールに同時書き込み
  DoubleWriteCacheStores https://github.com/hirocaster/double_write_cache_stores
- メンテナンス無しでプールの切り替えが可能。プールサイズの増強が可能に。

2プール構成のMemcached
入れ替え方法

通常時



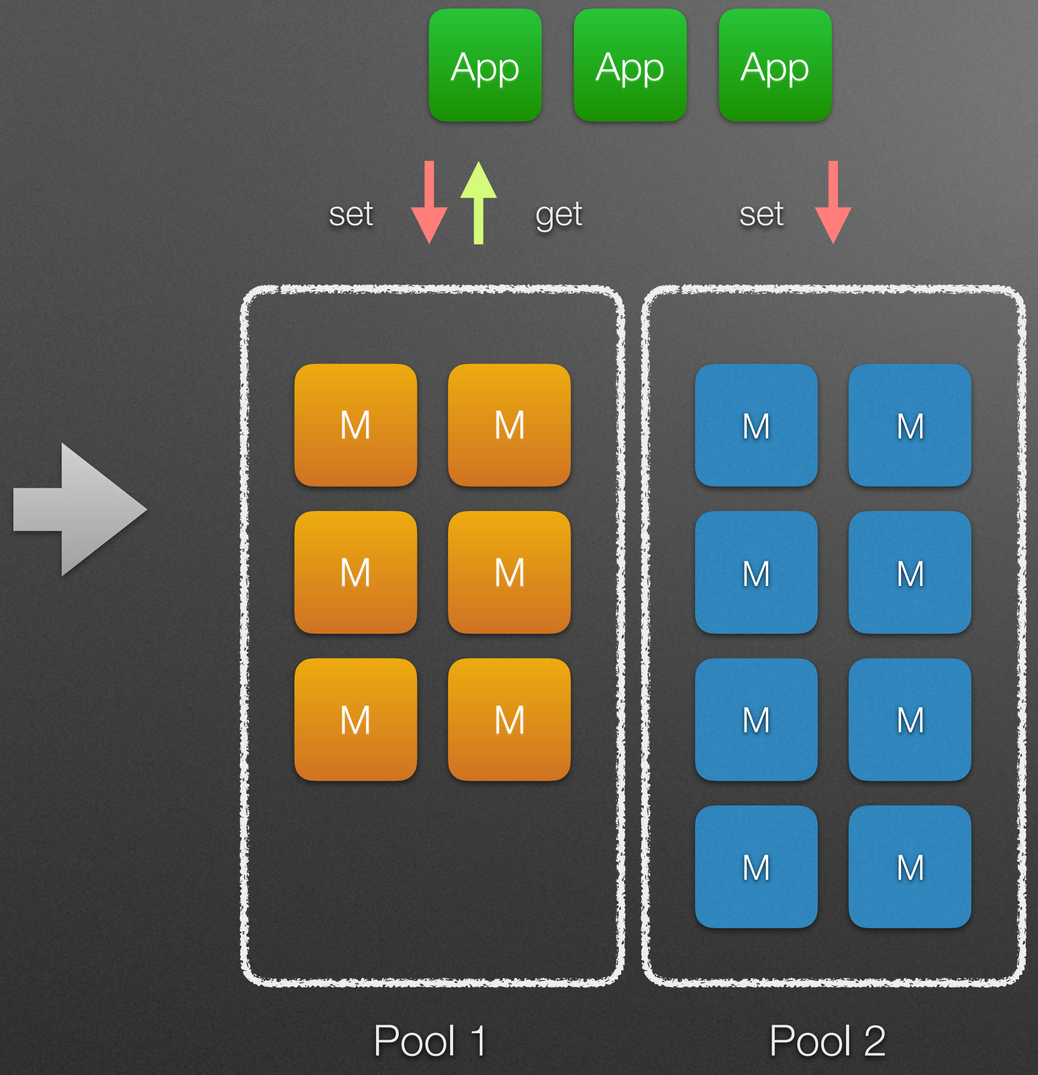
停止・増設



全クリアしておく

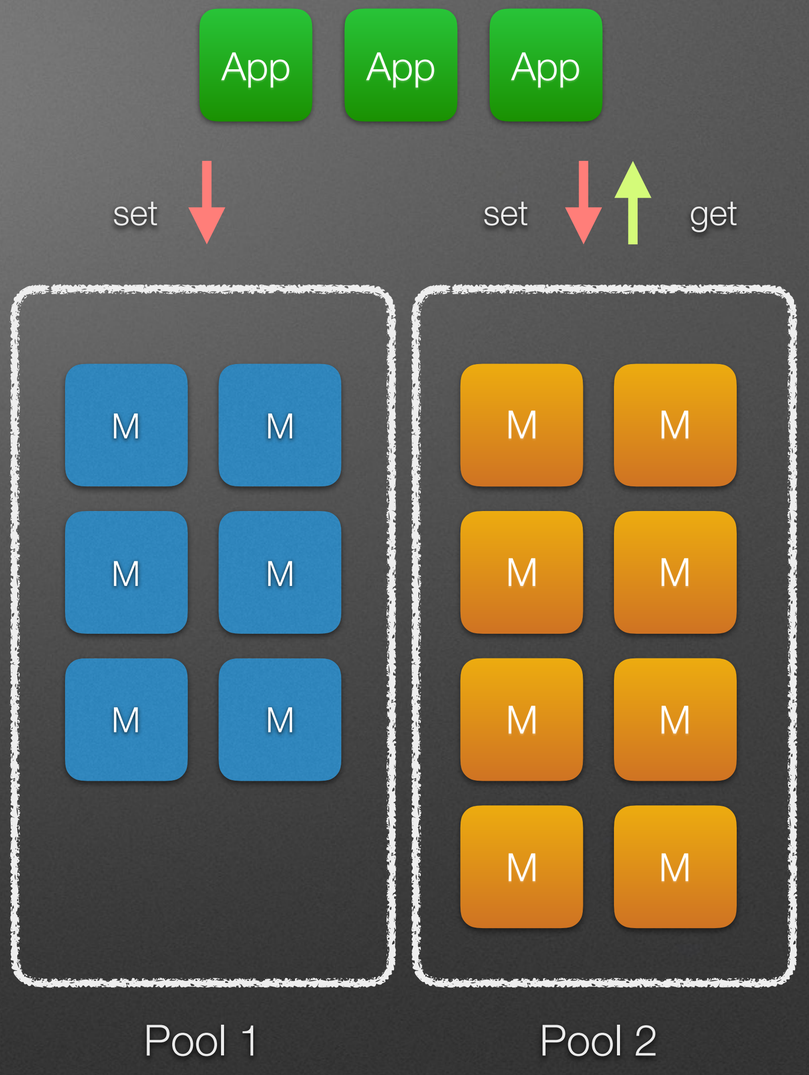
M Memcachedホスト

キャッシュあたため



しばらくあたためる

切り替え

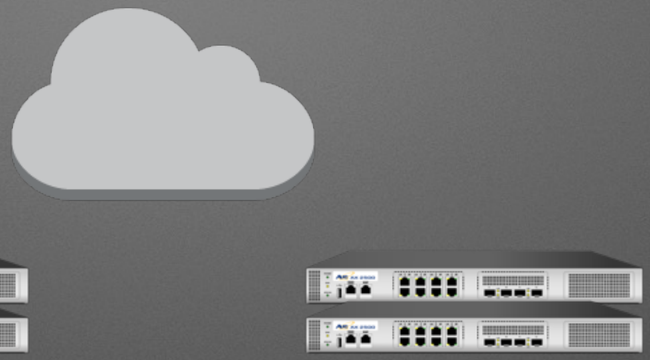


ロードバランサー

ロードバランサー

- Applicationサーバの前段はアプライアンスのロードバランサーを利用
- A10 Networks AX2500
- RESTful APIによる操作が可能
- 自社開発のGo製のCLIツール（a10-cli）
- Applicationサーバの追加、削除時にコマンド1つでサービスイン・アウトができる

Internet



A10 ロードバランサー

Applicationサーバ

nginx
unicorn

nginx
unicorn

nginx
unicorn

nginx
unicorn

MemcachedやMariaDBなど

モニタリング・監視

CloudForecast

- すべてのサーバのメトリクス取得
- 約1,000台のサーバが計測対象
- 様々な独自のメトリクスを追加

https://github.com/kazeburo/cloudforecast

GrowthForecast

- DCラックの電力値のグラフ化
- Fluentd Monitoring Agentのグラフ化

http://kazeburo.github.io/GrowthForecast/

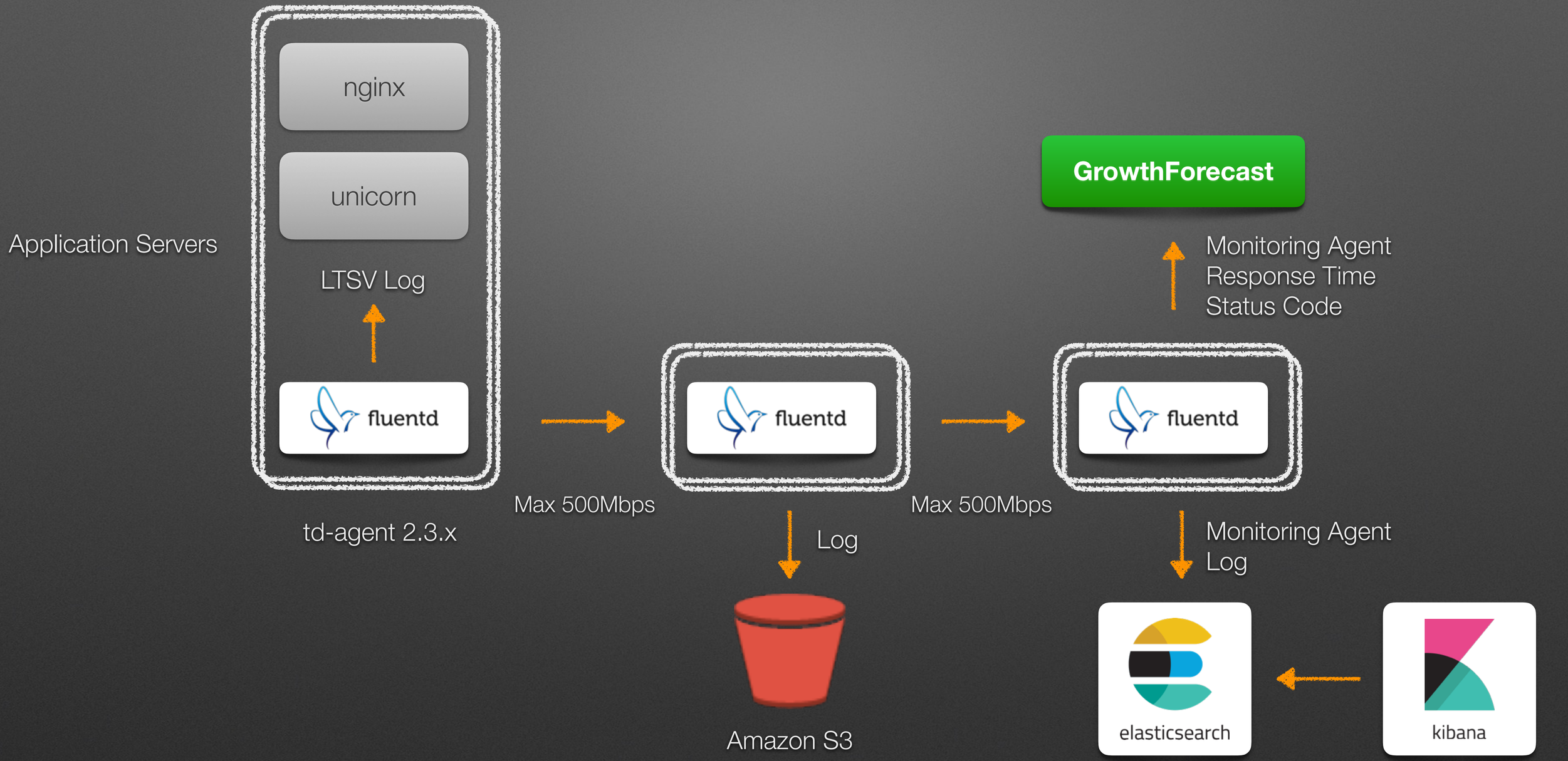
https://www.nagios.org/

- すべてのサーバの監視
- 12,000を超える監視項目
- 独自のプラグインを追加
- PagerDutyとの連携
  - 当番のスケジューリング
  - エスカレーションポリシー
- Slackへの通知

https://www.elastic.co/

- nginx, Applicationのログを
  Fluentdを使ってElasticsearchへ転送
- API毎の処理時間やレスポンスタイムの計測
- エラー頻度の計測
- 特定ホストからのアクセスの調査
- Slow Queryの可視化
- CloudTrailログの可視化

ログ転送フロー


Application Servers
nginx
unicorn
LTSV Log
fluentd
td-agent 2.3.x
Max 500Mbps
fluentd
Log
Amazon S3
Max 500Mbps
fluentd
GrowthForecast
Monitoring Agent
Response Time
Status Code
Monitoring Agent
Log
elasticsearch
kibana

日常のツール

- コミュニケーションはSlackで統一
- Hubot https://hubot.github.com 利用によるChatOpsの実現
- 開発環境へのデプロイなどをSlackから実行可能
- 詳しくはSoftware Design 2016年1月号のChatOps特集に
  http://gihyo.jp/magazine/SD/archive/2016/201601

- アプリケーションコードもインフラ構築用のCookbookなど
  すべてGitHub上で管理
- pull requestsベースの開発、レビュー
- GitHub上での活動はSlackへ通知される

- GitHubでpull requestsが作成されると自動テスト実行
- Dockerコンテナの生成、テスト実行、破棄
- Hubotとの連携

これからやっていきたいこと

- コードの改善、スケールアウト、キャッシュ利用など
  - スケールアップはできるだけやりたくない
  - 結果的にスケールダウンにつながるとベスト
  - これらは従来から継続して実施している
- ソフトウェアの継続的アップデート
  - Ubuntu Server 14.04 or 16.04 LTS（4/21リリース予定）への移行
  - Kernel、ミドルウェアのアップデートは積極的に実施

- MemcachedのSlab Rebalancing（ slab_reassign, slab_automove ）
  - まだ調査、検証段階なので運用事例があれば知りたい
- 監視、モニタリング環境の改善
  - 負荷や異常により素早く気づける運用に
  - もう少し自動化したいところも
  - Consulとの連携

- 社内の新規案件ではHashicorp社製ツール Terraformの利用が始まっている
  - モンストに後から導入するのはコストやリスクが高いので導入予定はなし
  - 部分的にツール利用による効率化はおこなっていきたい
- インフラテストの効率化
  - DockerやServerspecなどをさらに活用していきたい
- Chef Cookbookのリファクタリング、もっとシンプルに

最後に

エンジニア募集中！

http://xflag.com/recruit/career/engineer.html

ありがとうございました