Optimizing Bank-of-Anthos with Opsani and AppDynamics

Version 1.0 expires 30 August 2021

Access **: Public**

www.opsani.com

Bank of Anthos with AppDynamics

Bank of Anthos is a complete demo banking application that is K8s-based built with Java and Python components. Details on architecture and individual services are provided in Appendix 1. AppDynamics is an application performance management platform that allows for robust metric gathering and control of deployments. The following instructions explain how to instrument Bank of Anthos with AppDynamics agents.

Prerequisites and Components

1. `kubectl` cli command version appropriate to your cluster
2. bash command line interpreter (to run the deployment scripts)
3. target kubernetes cluster with at least 8 AWS m5.xl equivalent nodes available exclusively for this test
4. ability to create a namespace, or define a namespace in the `NAMESPACE` environment variable for Bank of Anthos
5. ability to create the metrics-service (or already have it installed) in the kube-systems administrative namespace
6. an AppDynamics account with sufficient licenses for all services (alternatively, selectively use the non-instrumented images for certain services accordingly), as well as proper allocation of these licenses under rules

Overview of Changes for use with AppDynamics

Bank of Anthos comprises 6 services (along with 2 databases). For use with AppDynamics, each of the 6 components has to be instrumented, which varies based on the underlying language of the component. Dynamic deployments (frontend and user-service) register a new AppD agent for each new pod initialized.

The Java components (balance-reader, ledger-writer, and transaction-history) utilize the AppDynamics Java machine agent, which is installed into the application's JVM. The following changes have been made:

1. An initContainer that pulls and installs the Java machine agent
2. Environment variables (both directly in the manifest and project-wide from the `config.yaml`) to set the AppDynamics parameters

3. An `/appdynamics` volume+mount to run the "`appd-env-script`" from `appd-scripts.yaml` and ensure all variables are passed correctly to the agent
4. An override entrypoint that starts the agent simultaneously with the component

The Python components (contacts, frontend and userservice) utilize the AppDynamics Python machine agent and have been modified as so:

1. Rebuild of the docker images contained within the `/src` to include the AppDynamics python package
2. Environment variables (both directly in the manifest and project-wide from the config.yaml) to set the AppDynamics parameters
3. An initContainer that runs the '`appd-pyagent`' script within `appd-scripts.yaml` to build an `appd.cfg` file
4. An override entrypoint to that preprends '`pyagent run`' to the `gunicorn` command based on the predefined entrypoint in the `/src/*/Dockerfile`

The `config.yaml` is the only file required to be modified with AppDynamics-specific parameters. General variables are stored in the "`appd-config`" ConfigMap, where `APPDYNAMICS_AGENT_APPLICATION_NAME` has to be set and all others will run with the default options. AppDynamics credentials are stored in "`appd-secrets`", and require the base-64 encoded username, account name, controller host name, password and access key.

QUICKSTART

Populate the `/kubernetes-manifests/config.yaml` and run the `/appdynamics/deploy.sh` script which will attempt to validate the pre-requisites, install any missing k8s service components, and install the Bank of Anthos application:

```
cd appdynamics && ./deploy.sh
```

Install Bank of Anthos (Manually)

The following commands will deploy the basic Bank-of-Anthos app, updated to use more realistic resources (requests/limits), and a load generator deployment that should drive 3-5 frontend pods to run and scale up to ~10 pods over time (Currently ~ 1 hour).

The following commands are run by the above `deploy.sh` script, but can be run manually instead.

Ensure kubectl is installed and pointing to your cluster

The following command will ensure that a) `kubectl` is installed, b) that you can talk to the cluster and c) that you see at least 6 nodes (the output should be a number).

```
kubectl get nodes | grep 'internal' | wc -l
```

Ensure you have a namespace and to simplify follow-on processes, that the namespace is default

```
echo "ensure NAMESPACE is an environment variable": export
NAMESPACE=${NAMESPACE:-bank-of-anthos-appdynamics} if [ "`kubectl create ns
${NAMESPACE} >& /dev/null; echo $?`" ]; then   echo `kubectl get ns ${NAMESPACE} |
grep ${NAMESPACE}` fi
```

Also, you may want to add the namespace to your kubeconfig as the "Default" for this cluster:

```
kubectl config set-context kubectl config get-contexts | awk '/^\*/ {print $2}'
--namespace ${NAMESPACE}
```

Ensure that the metrics service is installed and running

Metrics are needed to run the HPA pod autoscaler, and are simple point in time cpu and memory data captured from the kubelet service on each node. You need access to the kube-system namespace and the ability to create Cluster Roles and Cluster Role Bindings in order to apply this manifest from the Kubernetes metrics-sig:

```
kubectl apply -f
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components
.yaml
```

Finally, we can install Bank-of-Anthos Step one is to create a JWT token to support login via the Web UI with the following commands:

```
openssl genrsa -out jwtRS256.key 4096
openssl rsa -in jwtRS256.key -outform PEM -pubout -out jwtRS256.key.pub
kubectl create -n ${NAMESPACE} secret generic jwt-key --from-file=./jwtRS256.key
--from-file=./jwtRS256.key.pub rm ./jwtkey
```

Step two is to launch all of the manifests from the kubernetes-manifests directory which will complete the application deployment.

```
kubectl apply -n ${NAMESPACE} -f ../kubernetes-manifests/
```

Verify that the service is up and running

We can check to ensure that our pods are starting (this may take a moment):

```
kubectl -n ${NAMESPACE} get pods -w
```

This will wait and show you pods as they come online. type ^C (control-C) to quit watching for new pods.

Alternately, we can launch a port-forward to enable access to the Frontend service from our local machine:

```
kubectl port-forward -n ${NAMESPACE} svc/frontend 8080:http &
```

You should then be able to point to http://localhost:8080 and get a login page. The default user is testuser and the default password is password.

Load generation

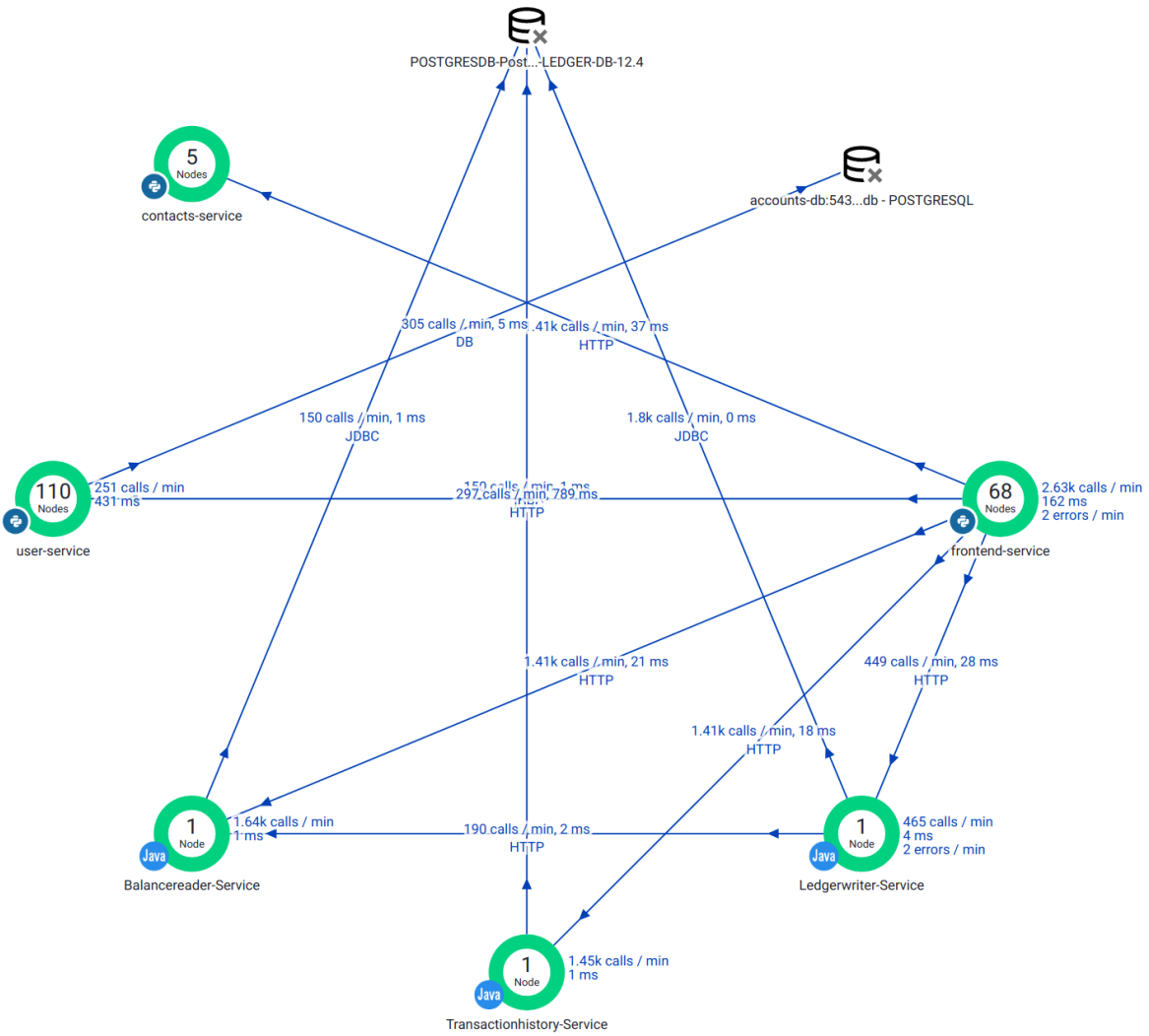
Load is automatically generated in a dynamic fashion with the loadgenerator pod. Opsani has modified this with the latest version of locust.io, and includes a dynamic sinusoidal load pattern. You can modify the parameters of the `kubernetes-manifests/loadgenerator.yaml` document with the following parameters:

- `STEP_SEC`: seconds per step, longer will generate a longer load range, usually `10` is good for initial tests, and `600` for longer term load.

- `USER_SCALE`: Number of users to vary, the more the heavier the load. `180` appears to be a good starting point for reasonable load.
- `SPAWN_RATE`: How quickly to change during the step, there is likely no need to change this parameter.
- `MIN_USERS`: As the sinusoidal shape varies between "`0`" and "`1`" multiplied by the `USER_SCALE` parameter, it is often good to ensure some load, we set this as `50` by default.

Examining in AppDynamics Console

At this point, the Bank of Anthos application should be running on your Kubernetes cluster, have dynamic load reaching it, and have AppDynamics agents running in each component that will be generating a flowmap and gathering metrics in the controller console.



(Optional) Install the Cluster Agent

The cluster agent allows for high-level insights into the underlying K8s behavior, such as:

- pod failures and restarts
- node starvation
- pod eviction threats and pod quota violations
- image and storage failures
- pending or stuck pods
- bad endpoints: detects broken links between pods and application components
- service endpoints in a failed state
- missing dependencies (Services, configMaps, Secrets)

To install the CA, update the `cluster-agent/cluster-agent.yaml` with the target appName, controllerUrl and account, then apply it along with the CA operator (Note: this is not deployed in the automatic `deploy.sh` script).

```
kubectl -n ${NAMESPACE} apply -f ../appdynamics/cluster-agent/
```

Uninstall Bank-of-Anthos

If a namespace was created for this project, the simplest approach is to simply delete the namespace.

Alternatively, clean out the deployed manifests:

```
kubectl delete -n ${NAMESPACE} -f ../kubernetes-manifests/
```

We will also want to clean up the manually deployed jwt-key secret:

```
kubectl delete secret jwt-key
```

And to really clean things out, you can delete the namespace and the metrics service as well:

```
kubectl delete ns ${NAMESPACE} kubectl apply -f
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components
.yaml
```

Opsani Servo for AppDynamics

Connector for Opsani Servo that utilizes AppDynamics agents to provide metrics for optimization. Either a standard RED (requests-error-duration) measurement set can be used, or a more specific Business Transactions (BT) set, which are used which map the end-to-end, cross-tier processing path used to fulfill a request for a service provided by the application. For example, in a fully-featured application such as Bank of Anthos, the "/payment" BT provides metrics for the full path that BT travels through, including both frontend-service and user-service. Note: due to the AppDynamics mapping process, BT-based optimizations can only be performed through the originating tier of the BT (frontend, in most cases).

Configuration

```
appdynamics:
  description: Update the app_id, tier, base_url and metrics to match your AppDynamics
configuration.
      Username, account and password set via K8s secrets
  app_id: appd-payment
  tier: frontend-service
  base_url: https://replaceme.saas.appdynamics.com
  metrics:
  - name: main_throughput
      unit: rpm
      query: Overall Application Performance|frontend-service|Individual
Nodes|frontend|Calls per Minute
```

Username, account and password credentials are set in the target kubernetes application via k8s secrets, and applied in the servo manifest as follows:

```
- name: SERVO_APPDYNAMICS_USERNAME
  valueFrom:
      secretKeyRef:
      name: appd-secrets
      key: username
- name: SERVO_APPDYNAMICS_ACCOUNT
  valueFrom:
      secretKeyRef:
      name: appd-secrets
      key: accountname
- name: SERVO_APPDYNAMICS_PASSWORD
  valueFrom:
      secretKeyRef:
      name: appd-secrets
      key: password
```

Usage

Latest image builds are available via `opsani/servox-appdynamics:edge`

Preconfigured metric templates are available via the Opsani console for both RED and BT optimizations.

Measurements

To differentiate measurements from a main and tuning set, metrics defined in the config are prepended with the respective set name. When measuring a business transaction, the BT is also appended. E.g. `main_throughput` along with `tuning_throughput`, or (in the case of a BT measurement) `main_payment_througphut` along with `tuning_payment_throughput`. Native aggregation occurs within the connector to identify and either average or sum metrics for all active nodes in the main set, and obtained directly in the case of the singleton tuning node. Additive metrics measured in `rpm` such as throughput are aggregated via a sum, and subsequently differentiated in the console to also provide the instance-sensitive value

(resulting in `main_throughput` and `main_throughput_pod`). Metrics measured in `ms` such as latency are returned already averaged to the pod count.

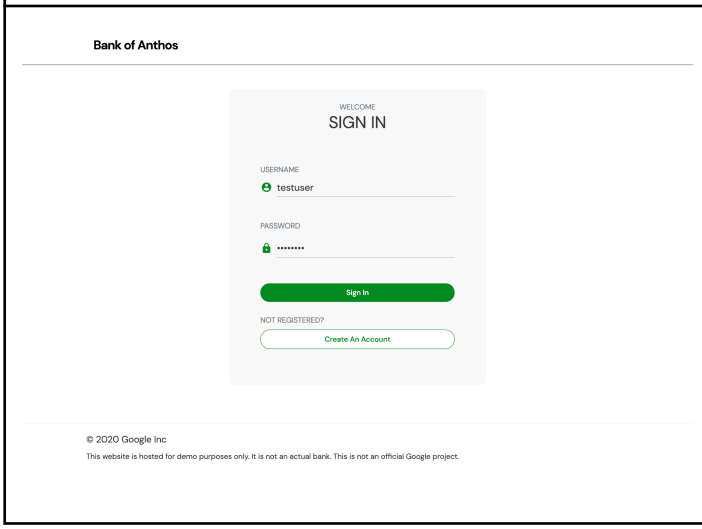
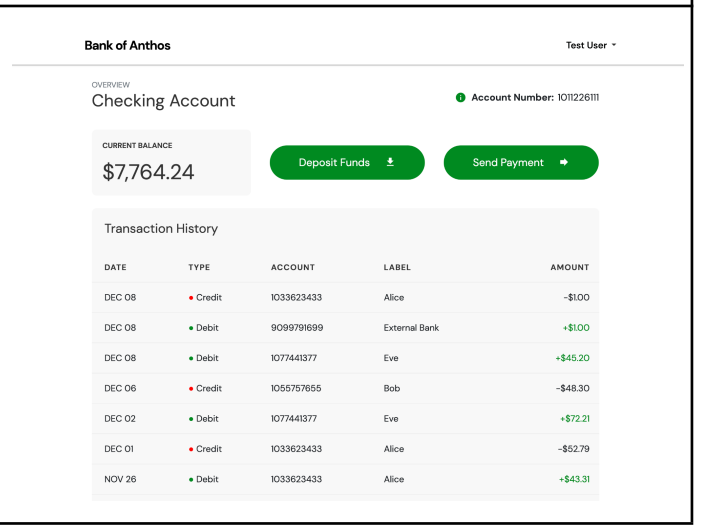
License

servo-appdynamics is distributed under the terms of the Apache 2.0 Open Source license. A copy of the license is provided in the LICENSE file at the root of the repository.

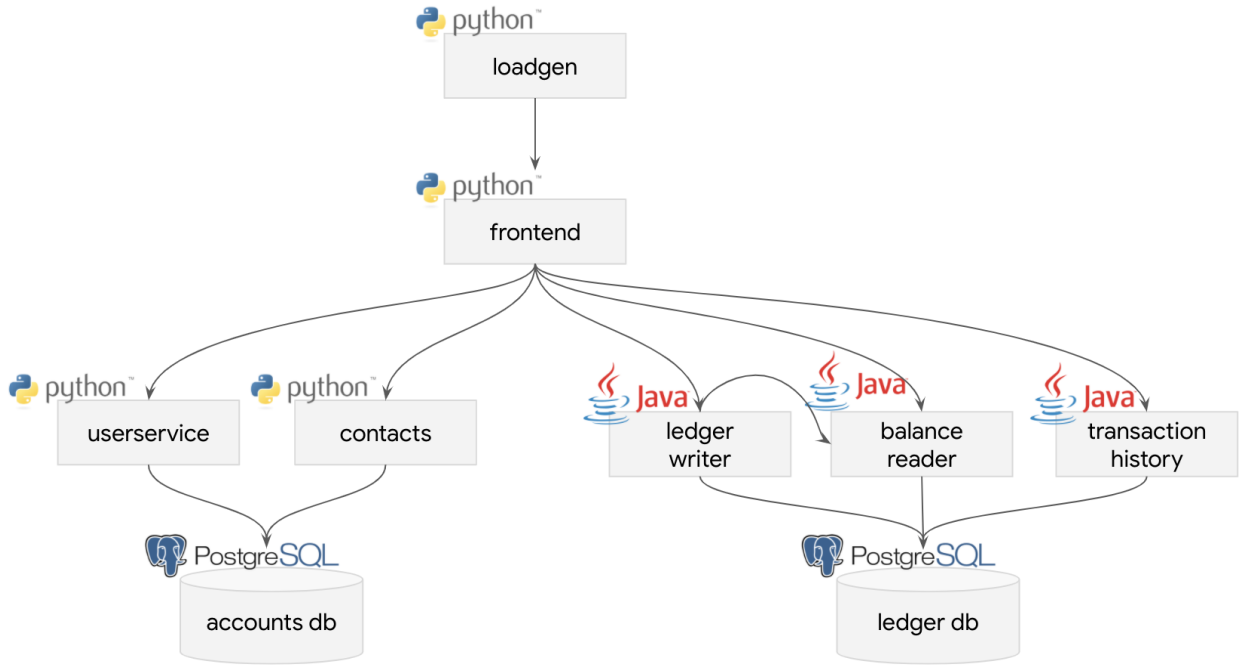
Appendix 1. The Bank of Anthos

The Bank of Anthos is a sample HTTP-based web app that simulates a bank's payment processing network, allowing users to create artificial bank accounts and complete transactions. Opsani uses this application to demonstrate a real polyglot application under optimization in a Kubernetes environment.

Screenshots

Sign In	Home
	

Service Architecture



Service	Language	Description
frontend	Python	Exposes an HTTP server to serve the website. Contains login page, signup page, and home page.
ledger-writer	Java	Accepts and validates incoming transactions before writing them to the ledger.
balance-reader	Java	Provides efficient readable cache of user balances, as read from ledger-db.
transaction-history	Java	Provides efficient readable cache of past transactions, as read from ledger-db.
ledger-db	PostgreSQL	Ledger of all transactions. Option to pre-populate with transactions for demo users.

user-service	Python	Manages user accounts and authentication. Signs JWTs used for authentication by other services.
contacts	Python	Stores list of other accounts associated with a user. Used for drop down in "Send Payment" and "Deposit" forms.
accounts-db	PostgreSQL	Database for user accounts and associated data. Option to pre-populate with demo users.
loadgenerator	Python/Locust	Continuously sends requests imitating users to the frontend. Periodically creates new accounts and simulates transactions between them.

Troubleshooting

See the Troubleshooting guide for resolving common problems.

Appendix 2. Bank of Anthos Quickstart With GKE

The following instructions describe getting Bank of Anthos running on Google Kubernetes Engine. If you already have a Google Cloud account you can start by opening a Cloud Shell.

1. **Create a Google Cloud Platform project** or use an existing project.

Set the `PROJECT_ID` environment variable and ensure the Google Kubernetes Engine API is enabled.

```
PROJECT_ID=""

gcloud services enable container --project ${PROJECT_ID}
```

2. **Clone this repository.**

```
git clone https://github.com/GoogleCloudPlatform/bank-of-anthos.git cd bank-of-anthos
```

3. **Create a GKE cluster.**

```
ZONE=us-central1-b

gcloud beta container clusters create bank-of-anthos \

--project=${PROJECT_ID} --zone=${ZONE} \

--machine-type=e2-standard-2 --num-nodes=4 \

--enable-stackdriver-kubernetes --subnetwork=default \

--tags=bank-of-anthos --labels csm=
```

4. **Deploy the demo JWT public key** to the cluster as a Secret. This key is used for user account creation and authentication.

```
kubectl apply -f ./extras/jwt/jwt-secret.yaml
```

5. **Deploy the sample app to the cluster.**

```
kubectl apply -f ./kubernetes-manifests
```

6. **Wait for the Pods to be ready.**

```
kubectl get pods
```

After a few minutes, you should see:

```
NAME                                 READY STATUS  RESTARTS  AGE
accounts-db-6f589464bc-6r7b7         1/1   Running 0         99s
balancereader-797bf6d7c5-8xvp6       1/1   Running 0         99s
contacts-769c4fb556-25pg2            1/1   Running 0         98s
frontend-7c96b54f6b-zkdbz            1/1   Running 0         98s
ledger-db-5b78474d4f-p6xcb           1/1   Running 0         98s
ledgerwriter-84bf44b95d-65mqf        1/1   Running 0         97s
loadgenerator-559667b6ff-4zsvb       1/1   Running 0         97s
transactionhistory-5569754896-z94cn 1/1   Running 0         97s
userservice-78dc876bff-pdhtl         1/1   Running 0         96s
```

7. **Access the web frontend in a browser** using the frontend's `EXTERNAL_IP`.

```
kubectl get service frontend | awk '{print $4}'
```

Example output - **do not copy**.

```
EXTERNAL-IP

35.223.69.29
```