



Web



Monitoring



WEB &gt; GUIDES &gt; PSPDFKIT SERVER &gt; MONITORING

# Enable metrics export



PSPDFKit Server has been deprecated and replaced by [Document Engine](#). To migrate to Document Engine and unlock advanced document processing capabilities, refer to our [migration guide](#). Learn more about these enhancements on our [blog](#).

Similar to how it works with [logs](#), Docker allows you to [gather system-level metrics](#) about running containers, like CPU usage, memory consumption, block device IO, etc. AWS, Google Cloud Platform, and Azure all offer solutions to collect these metrics from containers launched on their respective infrastructures.

See the following:

- ✧ [Using Container Insights](#)
- ✧ [Cloud Operations for GKE](#)
- ✧ [Enable Azure Monitor for containers](#)

Since version 2020.5.0, Server provides the capability to send internal metrics to any metrics engine supporting the [DogStatsD protocol](#), which is an extension of the popular [StatsD protocol](#). Internal metrics offer more fine-grained insights into Server performance and help to pinpoint specific issues. Check out our [metrics integration](#) section below on how to enable internal Server metrics when running on-premises or in the cloud. The list of all exported metrics is available [here](#).

## Metrics integration



ASK AI

PSPDFKit Server sends metrics using an open [DogStatsD protocol](#), which is an extension of the popular [StatsD protocol](#). Any metric collection engine that understands this protocol can ingest metrics exported by Server — including [Telegraf](#), [AWS CloudWatch Agent](#), and [DogStatsD](#) itself.

To enable exporting metrics, set the `STATSD_HOST` and `STATSD_PORT` configuration variables to point to the hostname and port where the collection engine is running.

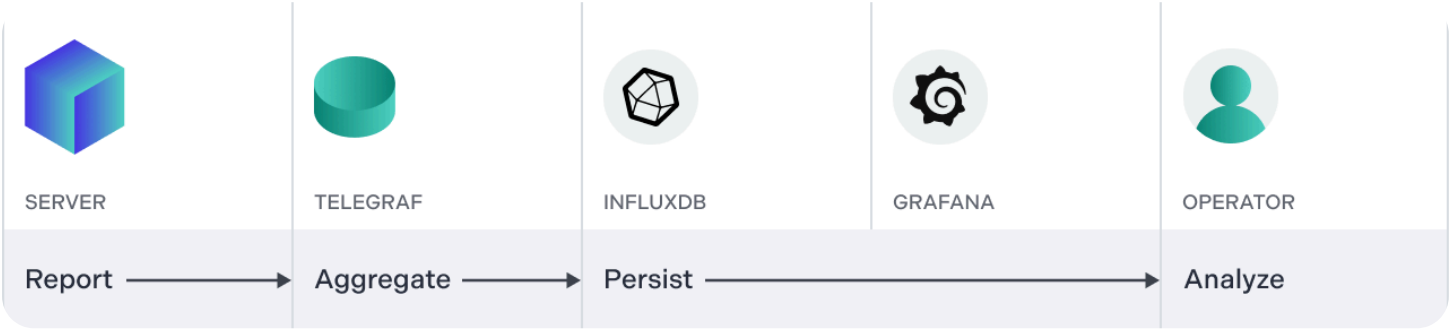
You can also define custom metric tags by setting the `STATSD_CUSTOM_TAGS` environment variable to a comma-separated list of key-value pairs (with key and value separated by a `=`). For example, if you want to tag metrics with a region and a role, you can set the environment variable to `region=eu,role=viewer`.

This guide covers instructions on how to integrate Server metrics with various metric collection systems, more specifically:

- ❖ [Docker-based setup with Telegraf, InfluxDB, and Grafana](#)
- ❖ [AWS CloudWatch](#)
- ❖ [Google Cloud Monitoring](#)
- ❖ [Azure Monitor](#)

## Docker-based setup with Telegraf, InfluxDB, and Grafana

This deployment scenario integrates Server with [Telegraf](#), [InfluxDB](#), and [Grafana](#) via `docker-compose`. This approach is a great fit when you want to maintain complete control over the entire environment, you can't deploy in the cloud, or you just want to try things out. It can be also adapted to other deployment orchestration tools based on Docker containers, such as [Kubernetes](#).



In this setup, Server sends metrics to Telegraf, which aggregates time during fixed-period time buckets. After metrics are aggregated, they're sent to InfluxDB for storage. Finally, the operator can view and analyze metrics in the Grafana dashboard by writing queries for InfluxDB.

# Prerequisites

In order to complete this section, you'll need to have [Docker](#) with `docker-compose` installed. You'll also need a PSPDFKit Server activation key or a trial license key; refer to the [Product Activation](#) guide.

## Setting up

To get started, clone the repository with the configuration files: <https://github.com/PSPDFKit/pspdfkit-server-example-metrics>. In the root of the repository, you'll find the following `docker-compose.yml` file:

```
1  version: "3.8"
2
3  services:
4    grafana:
5      image: grafana/grafana:7.1.5
6      ports:
7        - 3000:3000
8      environment:
9        - GF_SECURITY_ADMIN_PASSWORD=secret
10     depends_on:
11       - influxdb
12     volumes:
13       - ./grafana/provisioning/datasources:/etc/grafana/provisioning/datasources
14       - ./grafana/provisioning/dashboards:/etc/grafana/provisioning/dashboards
15       - ./grafana/dashboards:/var/lib/grafana/dashboards:ro
16   influxdb:
17     image: influxdb:1.8.2
18   telegraf:
19     image: telegraf:1.14.5-alpine
20     depends_on:
21       - influxdb
22     volumes:
23       - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf:ro
24   db:
25     image: postgres:15
26     environment:
27       POSTGRES_USER: pspdfkit
28       POSTGRES_PASSWORD: password
29       POSTGRES_DB: pspdfkit
30       PGDATA: /var/lib/postgresql/data/pgdata
31     volumes:
32       - pgdata:/var/lib/postgresql/data
33   pspdfkit:
34     image: pspdfkit/pspdfkit:2024.1.2
35     environment:
36       STATSD_HOST: telegraf
37       STATSD_PORT: 8125
38
39     ACTIVATION_KEY: <YOUR_ACTIVATION_KEY>
40     DASHBOARD_USERNAME: dashboard
```

```

41     DASHBOARD_PASSWORD: secret
42
43     PGUSER: pspdfkit
44     PGPASSWORD: password
45     PGDATABASE: pspdfkit
46     PGHOST: db
47     PGPORT: 5432
48     API_AUTH_TOKEN: secret
49     SECRET_KEY_BASE: secret-key-base
50     JWT_PUBLIC_KEY: |
51         -----BEGIN PUBLIC KEY-----
52         MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2gzhmJ9TDanEzWdPlWG+
53         0Ecwbe7f3bv6e5UUpvcT5q68IQJKP47AQdBAnSlFVi4X9SaurbWoXdS6jpmPpk24
54         QvitzLNFphHdwjFBeItAOa6taZrSusoFvrtK9x5xsW4zzt/bkpUraNx82Z8MwLwr
55         t6HlY7dgO9+xBaAbj4tld2t+0HS8O/ed3CB6T2lj6S8AbLDSEFc9ScO6Uc1XJlSo
56         rgyJJSPCpNhSq3AubEZlwMSliEtgAzTPRDsQv50qWIbn634HLWxTP/UH6YNJBwzt
57         3O6q29kTtjXlMGXCvin37PyX4Jy1IiPFwJm45aWJGKSfVGMDojTJbuUtM+8P9Rrn
58         AwIDAQAB
59         -----END PUBLIC KEY-----
60     JWT_ALGORITHM: RS256
61     ports:
62     - 5000:5000
63     depends_on:
64     - db
65     - telegraf
66
67     volumes: pgdata:

```

This configuration file defines all services required to deploy Server and observe metrics reported by it. Remember to swap out the `<YOUR_ACTIVATION_KEY>` placeholder with the actual activation key of PSPDFKit Server. Note that Server is configured to send metrics to Telegraf: `STATSD_HOST` points to `telegraf` service, and `STATSD_PORT` is configured to use port 8125.

In the `telegraf/` subdirectory, you'll find the configuration for the Telegraf agent, `telegraf.conf`:

```

1  [agent]
2    interval = "5s"
3
4  [[inputs.statsd]]
5    service_address = ":8125"
6
7    metric_separator = "."
8    datadog_extensions = true
9
10  templates = [
11    "*.measurement.field",
12    ".*.measurement.measurement.field",
13  ]
14
15  [[outputs.influxdb]]

```

```
16 urls = ["http://influxdb:8086"]
17 database = "pspdfkit"
```

Let's go through the options defined here:

- ❖ `interval` specifies how often Telegraf takes all the received data points and aggregates them into metrics.
- ❖ `inputs.statsd.service_address` defines the address of the UDP listener. The port number here must match the `STATSD_PORT` configuration for Server.
- ❖ `inputs.statsd.metric_separator` needs to be set to `.` when used with PSPDFKit Server.
- ❖ `inputs.statsd.datadog_extensions` needs to be set to `true` so that metric tags sent by Server are parsed correctly.
- ❖ `inputs.statsd.templates` defines how names of metrics sent by Server are mapped to Telegraf's metric representation. This needs to be set to the exact value shown in the configuration file above.
- ❖ `outputs.influxdb` is the URL of the InfluxDB instance where data will be stored.
- ❖ `outputs.influxdb.database` is the name of the database in InfluxDB where metrics will be saved. You'll need to use the same name when querying data in Grafana.


The above is just an example configuration file, and it's by no means a complete configuration suitable for production deployments; refer to the [Telegraf documentation](#) for all the available options.

In the `grafana/` subdirectory, you'll find definitions of data sources, along with an example dashboard. You can provide all this data through Grafana UI; this is just an example to get you up and running quickly. Refer to the [Grafana documentation](#) for more information.

Now when you run `docker-compose up` from the directory when the `docker-compose.yml` file is placed, all the components will be started.

## Viewing metrics

Head over to `http://localhost:3000` in your browser to access the Grafana dashboard using admin/secret credentials to log in. Now open the PSPDFKit Server dashboard. You'll see a dashboard like in the image below, but most likely with different data points.

 **Note:** If no data is shown in the dashboard, open the Server dashboard at `<SERVER_URL>/dashboard`, upload a couple documents, and perform some operations on them.

The dashboard shows a few crucial Server metrics — you can see their definitions by selecting a panel and choosing Edit. For a complete reference of available metrics, see [this guide](#).

## AWS CloudWatch

If you're deploying your services to AWS, chances are you're already monitoring them using AWS CloudWatch. If you followed our [AWS deployment guide](#) and host Server on AWS ECS, system-level metrics like CPU and memory utilization are automatically collected for you. This section shows how you can integrate PSPDFKit Server with the [CloudWatch Agent](#) to export internal Server metrics to CloudWatch.

## Prerequisites

This section assumes you followed the [Server AWS deployment guide](#) or have deployed PSPDFKit Server to AWS ECS backed by an EC2 instance yourself.

## Setting up

First, you'll need to install the CloudWatch agent on the host where it's reachable by PSPDFKit Server. Refer to the [CloudWatch agent installation guide](#) for specific instructions.

The next step is agent configuration:

```
1 {
2   "metrics": {
3     "namespace": "PSPDFKitServer",
4     "metrics_collected": {
5       "statsd": {
6         "service_address": ":8125"
7       }
8     }
9   }
10 }
```

This configuration specifies that the agent should start a StatsD-compatible listener on port 8125. In addition, all metrics collected by the agent will be placed under the `PSPDFKitServer` namespace. Save the configuration in the `cwagent.json` file on the host where you installed the agent and run:

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-co
```

**Note:** The command above requires root privileges.

You can see that the agent is up by running the following:

```
systemctl status amazon-cloudwatch-agent.service
```

The only thing that's left is to configure PSPDFKit Server to send metrics to the agent. You'll need the IP address or the DNS entry for the host — this depends on where you deployed the agent. Make sure Server can reach the agent on port 8125 (e.g. if you deployed it to AWS EC2, configure the security group to allow incoming UDP traffic on port 8125).

Now head to the [ECS task definitions](#) page and modify the task definition for PSPDFKit Server. Click on the most recent revision, select Create new revision, and scroll down to edit the Server container. Set the `STATSD_HOST` environment variable to point to the CloudWatch agent's IP address or DNS entry, and set `STATSD_PORT` to point to 8125.

STATSD_HOST	Value ▼	<CW_AGENT_IP_OR_DNS> ✕
STATSD_PORT	Value ▼	8125 ✕
Add key	Value ▼	Add value

Save the changes and create a new revision. Now update the ECS service that's running the PSPDFKit Server task and change the task definition revision to the one you just created. Then wait until the task restarts.

## Viewing metrics

You can now go to the AWS CloudWatch console to view the metrics exported by Server. (Note that it takes a while until the agent sends the collected metrics upstream). Use this link to go straight to the

`PSPDFKitServer` namespace in the metrics browser:

[https://console.aws.amazon.com/cloudwatch/home#metricsV2:graph=~\(\)  
\(\);namespace=~'PSPDFKitServer'](https://console.aws.amazon.com/cloudwatch/home#metricsV2:graph=~();namespace=~'PSPDFKitServer').

As an example, to view the HTTP response time metric, search for `http_server_req_end` in the search box and tick all the checkboxes. You'll see a graph of HTTP timing metrics grouped by the HTTP method and response status code.



The data points you'll see will most likely be different than what's shown in the screenshot above.

**Note:** If there's no data shown in the dashboard, open the Server dashboard at `<SERVER_URL>/dashboard`, upload a couple documents, and perform some operations on them.

You can use the CloudWatch console to browse the available metrics. For more advanced use, see the CloudWatch [search expressions](#) and [metric math](#) guides. Check out the complete list of metrics exported by PSPDFKit Server on the [metrics reference page](#).

## Google Cloud Monitoring

[Google Cloud Monitoring](#) (formerly known as Stackdriver) is a monitoring service provided by the Google Cloud Platform. It's a great choice when you deploy your services on the Google Kubernetes Engine (GKE) stack, since it provides metrics for all the [Kubernetes](#) resources out of the box. To forward metrics from PSPDFKit Server to Google Cloud Monitoring, you'll use [Telegraf](#).

## Prerequisites

To follow this section, make sure you've deployed PSPDFKit Server to [GKE](#) as outlined in our deployment guide.

## Setting up

To get started, first you need to expose the Telegraf agent configuration as a [ConfigMap](#) in the GKE cluster. Save the following ConfigMap definition in the `telegraf-config.yml` file:

```
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: telegraf-config
5  data:
```



```

6   telegraf.conf: |
7       [agent]
8         interval = "90s"
9         flush_interval = "90s"
10
11
12     [[inputs.statsd]]
13         service_address = ":8125"
14
15         metric_separator = "."
16         datadog_extensions = true
17
18         templates = [
19             ".* measurement.field",
20             ".*.* measurement.measurement.field",
21         ]
22
23     [[outputs.stackdriver]]
24         project = "<GCP PROJECT ID>"
25         namespace = "pspdfkit"

```

This ConfigMap embeds the Telegraf configuration file directly. Let's go through the options defined here:

- ❖ `interval` and `flush_interval` specify how often the metrics are aggregated and how often they're pushed to Google Cloud. Make sure `flush_interval` isn't less than the `interval`, and that `interval` is set to at least 60 seconds. This is because the Google Cloud Monitoring API allows you to create, at most, one data point per minute.
- ❖ `inputs.statsd.service_address` defines the address of the UDP listener. Use the port number defined here when creating a [Service](#) for the Telegraf agents.
- ❖ `inputs.statsd.metric_separator` needs to be set to `.` when used with PSPDFKit Server.
- ❖ `inputs.statsd.datadog_extensions` needs to be set to `true` so that metric tags sent by Server are parsed correctly.
- ❖ `inputs.statsd.templates` defines how names of metrics sent by Server are mapped to Telegraf's metric representation. This needs to be set to the exact value shown in the configuration file above.
- ❖ `outputs.stackdriver` is a configuration of the Google Cloud Monitoring output plugin. Make sure to put your actual GCP project name here.

The above is just an example configuration file, and it's by no means a complete configuration suitable for production deployments; refer to the [Telegraf documentation](#) for all the available options.

You can create the ConfigMap in the cluster by running:

```
kubectl apply -f telegraf-config.yml
```



Now that the configuration is available, deploy Telegraf itself:

```
1  apiVersion: apps/v1
2  kind: DaemonSet
3  metadata:
4    name: telegraf
5  spec:
6    template:
7      metadata:
8        labels:
9          app: telegraf
10     spec:
11       containers:
12         - name: telegraf
13           image: telegraf
14           volumeMounts:
15             - name: telegraf-config
16               mountPath: /etc/telegraf/
17               readOnly: true
18       volumes:
19         - name: telegraf-config
20           configMap:
21             name: telegraf-config
22             items:
23               - key: telegraf.conf
24                 path: telegraf.conf
25 ---
26 apiVersion: v1
27 kind: Service
28 metadata:
29   name: telegraf
30 spec:
31   clusterIP: None
32   selector:
33     app: telegraf
34   ports:
35     - protocol: UDP
36       port: 8125
37       targetPort: 8125
```



This YAML file defines the Telegraf `DaemonSet`, which deploys a single Telegraf agent to every node in the GKE cluster, along with a headless Service that allows Server to reach Telegraf using a domain name. Notice that you're mounting the previously defined Telegraf configuration as a file in the Telegraf container. Save that definition in `telegraf.yml` and run:

```
kubectl apply -f telegraf.yml
```



The last thing you need to do is configure PSPDFKit Server to send metrics to Telegraf. Modify the Server resource definition file by adding these two environment variables:

```
1  env:
2    ...
3    - name: STATSD_HOST
4      value: telegraf
5    - name: STATSD_PORT
6      value: "8125"
```



Make sure to recreate the Server deployment by running `kubectl apply -f` on the file where it's defined.

## Viewing metrics

To view metrics, head over to Monitoring in the [Google Cloud console](#). If you follow that link, you'll land on the Metrics Explorer page, where you can search for all the PSPDFKit Server metrics. Note that the PSPDFKit Server metrics are available under the Global resource.

For example, search for `pspdfkit/http_server/req_end_mean` to view the average HTTP response time of PSPDFKit Server. Apply a filter to only view metrics with a `standard` group, and group them by the HTTP method.

The data points you'll see will most likely be different than what's shown in the screenshot above.

**Note:** If there's no data shown in the dashboard, open the Server dashboard at `<SERVER_URL>/dashboard`, upload a couple documents, and perform some operations on them.

You can now add the chart to the dashboard, or continue exploring [available metrics](#). Learn more about how to build an advanced monitoring solution on top of Google Cloud Monitoring by following the [relevant guides](#).

## Azure Monitor

If you're already using tools from the Microsoft ecosystem and deploying your software to Azure, you should consider sending metrics to [Azure Monitor](#). As with other metrics aggregation services available natively on cloud platforms, it provides a wide range of metrics for resources on the platform, and it can also be used to store and visualize custom metrics from any application, including PSPDFKit Server. You'll use Telegraf to export metrics from PSPDFKit Server to Azure Monitor.

## Prerequisites

In order to follow this section, make sure you've deployed PSPDFKit Server to [AKS](#).

## Setting up

Sending metrics to Azure Monitor requires authentication — any service that wants to publish metrics needs to have specific permissions. The easiest way to do this is to assign a built-in Monitoring Metrics Publisher role to the [managed identity](#) sending metrics. However, at the moment of writing, Azure doesn't provide a native way to assign managed identities to [Kubernetes Pods](#) running on AKS. In order to assign required permissions to pods running Telegraf agent, you'll deploy the [AAD Pod Identity](#) operator (an open source project built by the Azure team) to bind managed identities to relevant pods.

## Deploying the AAD pod identity operator

As a first step, you must grant the [service principal](#) or [managed identity](#) that runs your AKS cluster nodes permission to assign identities to those nodes. Follow [this document](#) to find out if you're running AKS nodes with service principal or managed identity, and if so, to learn how to obtain the service principal or managed identity client ID.

After you get the ID, make sure to save it in the `CLUSTER_IDENTITY_CLIENT_ID` environment variable.

Now, run the following commands:

```
1 SUBSCRIPTION_ID=$(az account show --query id -o tsv)
2 CLUSTER_RESOURCE_GROUP_NAME=$(az aks show --resource-group pspdfkitresourcegroup --query resourceGroup -o tsv)
3 CLUSTER_RESOURCE_GROUP="/subscriptions/${SUBSCRIPTION_ID}/resourcegroups/${CLUSTER_RESOURCE_GROUP_NAME}"
4 # Assign permission to modify properties of the node VMs in the cluster.
5 az role assignment create --role "Virtual Machine Contributor" --assignee ${CLUSTER_IDENTITY_CLIENT_ID} --scope $CLUSTER_RESOURCE_GROUP
6 # Assign permission to assign identities created in the cluster resource group
7 az role assignment create --role "Managed Identity Operator" --assignee ${CLUSTER_IDENTITY_CLIENT_ID} --scope $CLUSTER_RESOURCE_GROUP
```

**Note:** If you get an error that says you have insufficient permissions, contact your Azure Active Directory (Azure AD) administrator.

After the permissions have been assigned, you can deploy the operator. Run the following in the shell:

```
1 kubectl apply -f https://raw.githubusercontent.com/Azure/aad-pod-identity/master
2 kubectl apply -f https://raw.githubusercontent.com/Azure/aad-pod-identity/master
```

When you run `kubectl get deployments.apps`, you should see `mic` deployment up and running:

```
1 NAME          READY   UP-TO-DATE   AVAILABLE   AGE
2 mic           1/1     1             1           3h56m
3 pspdfkit      1/1     1             1           93m
```

## Creating identity with permissions to publish metrics

Now you're going to create an identity with permissions to write metrics to Azure Monitor. You'll later assign this to the pod running Telegraf agent.

First, create a new identity:

```
1 az identity create --name telegraf --resource-group ${CLUSTER_RESOURCE_GROUP}
2 TELEGRAF_IDENTITY_ID=$(az identity show --resource-group ${CLUSTER_RESOURCE_GROUP} --name telegraf --query id -o tsv)
3 TELEGRAF_IDENTITY_CLIENT_ID=$(az identity show --resource-group ${CLUSTER_RESOURCE_GROUP} --name telegraf --query clientId -o tsv)
```

Then assign it a Monitoring Metrics Publisher role:

```
1 CLUSTER_ID=$(az aks show --resource-group pspdfkitresourcegroup --name pspdfkit --query id -o tsv)
2 az role assignment create --role 'Monitoring Metrics Publisher' --assignee $TELEGRAF_IDENTITY_CLIENT_ID --scope $CLUSTER_ID
```

Note that the identity only has permission to write metrics in the scope of the AKS cluster.

Now that the identity is created, you need to let the AAD Pod Identity operator know that you want to assign that identity to specific nodes in the cluster. You can do this by creating `AzureIdentity` and `AzureIdentityBinding` custom resources:

```
1 apiVersion: aadpodidentity.k8s.io/v1
2 kind: AzureIdentity
3 metadata:
4   name: telegraf
5 spec:
6   type: 0
```

```

7   resourceID: <TELEGRAF_IDENTITY_ID>
8   clientID: <TELEGRAF_IDENTITY_CLIENT_ID>
9   ---
10  apiVersion: aadpodidentity.k8s.io/v1
11  kind: AzureIdentityBinding
12  metadata:
13    name: telegraf-binding
14  spec:
15    azureIdentity: telegraf
16    selector: telegraf

```

Copy the YAML definition above, and replace `<TELEGRAF_IDENTITY_ID>` and `<TELEGRAF_IDENTITY_CLIENT_ID>` with the values of `$TELEGRAF_IDENTITY_ID` and `$TELEGRAF_IDENTITY_CLIENT_ID` environment variables, respectively. This definition means that the `telegraf` identity you created ( `azureIdentity: telegraf` ) should be assigned to any pod whose `aadpodidbinding` label matches the value of `telegraf` ( `selector: telegraf` ).

Create the resource in the cluster by running `kubectl apply -f telegraf-identity.yml` .

## Configuring Telegraf

Since the identity for Telegraf is ready, you can prepare its configuration. You'll provision the configuration file in Kubernetes' `ConfigMap` :

```

1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: telegraf-config
5  data:
6    telegraf.conf: |
7      [agent]
8        interval = "60s"
9        flush_interval = "60s"
10
11     [[inputs.statsd]]
12       service_address = ":8125"
13
14       percentiles = [90]
15
16       metric_separator = "."
17       datadog_extensions = true
18
19     templates = [
20       ".* measurement.field",
21       ".*.* measurement.measurement.field",
22     ]
23
24     [[outputs.azure_monitor]]

```



```
25     namespace_prefix = "pspdfkit/"
26     resource_id = <CLUSTER_ID>
```

Replace the `<CLUSTER_ID>` placeholder with the value of the `$CLUSTER_ID` environment variable and save the configuration in the `telegraf-config.yml` file.

Let's quickly discuss the configuration options:

- ❖ `interval` and `flush_interval` specify how often the metrics are collected and how often they're written to Azure Monitor. Make sure `flush_interval` isn't less than the `interval`.
- ❖ `inputs.statsd.service_address` defines the address of the UDP listener. Use the port number defined here when creating a [Service](#) for the Telegraf agents.
- ❖ `inputs.statsd.metric_separator` needs to be set to `.` when used with PSPDFKit Server.
- ❖ `inputs.statsd.datadog_extensions` needs to be set to `true` so that metric tags sent by Server are parsed correctly.
- ❖ `inputs.statsd.templates` defines how names of metrics sent by Server are mapped to Telegraf's metric representation. This needs to be set to the exact value shown in the configuration file above.
- ❖ `outputs.azure_monitor` is a configuration for the Azure Monitor exporter. The `resource_id` is the cluster ID you used when granting metric publishing privileges to the Telegraf identity.

The above is just an example configuration file, and it's by no means a complete configuration suitable for production deployments; refer to the [Telegraf documentation](#) for all the available options.

Create the `ConfigMap` in the cluster by running:

```
kubectl apply -f telegraf-config.yml
```



## Deploying Telegraf and configuring PSPDFKit Server

You can now deploy Telegraf to the cluster, instructing it to use the provisioned configuration. The following file defines a `DaemonSet` and a [headless Service](#) for the Telegraf agents:

```
1  apiVersion: apps/v1
2  kind: DaemonSet
3  metadata:
4    name: telegraf
5  spec:
```



```

6   selector:
7     matchLabels:
8       app: telegraf
9   template:
10    metadata:
11      labels:
12        app: telegraf
13        aadpodidbinding: telegraf
14    spec:
15      containers:
16        - name: telegraf
17          image: telegraf
18          volumeMounts:
19            - name: telegraf-config
20              mountPath: /etc/telegraf/
21              readOnly: true
22      volumes:
23        - name: telegraf-config
24          configMap:
25            name: telegraf-config
26            items:
27              - key: telegraf.conf
28                path: telegraf.conf
29 ---
30 apiVersion: v1
31 kind: Service
32 metadata:
33   name: telegraf
34 spec:
35   clusterIP: None
36   selector:
37     app: telegraf
38   ports:
39     - protocol: UDP
40       port: 8125
41       targetPort: 8125

```

The `DaemonSet` will make sure there's a single Telegraf agent available on each node in the cluster, and the headless service allows communication with the agent using a domain name. Notice that `aadpodidbinding` is set to `telegraf`, so that the identity binding you created before gets applied and the Telegraf agent will be granted permission to write metrics to Azure Monitor. Save the specification above in the `telegraf.yml` file and run `kubectl apply -f telegraf.yml` to trigger deployment.

Finally, the last part is to configure PSPDFKit Server to start sending metrics to the Telegraf agent. Edit the Server deployment file and add these two environment variables:

```

1  env:
2    ...
3    - name: STATSD_HOST

```



```
4     value: telegraf
5   - name: STATSD_PORT
6     value: "8125"
```


Update the Server deployment by running `kubectl apply -f` on the deployment definition file.

## Viewing metrics

In order to view metrics, go to Azure Monitor service in [Azure portal](#) and navigate to [Metrics](#) in the sidebar. You'll be prompted to select a monitored resource — since you used the AKS cluster as the resource for which the metrics are written, you need to find the cluster by typing its name in the search field, select the checkbox, and click Apply.

In the Metric Namespace selection box, find `pspdfkit/http_server` and choose `req_end_90_percentile` for the metrics. Pick Max as the aggregation. Now click Add filter at the top and keep only those metrics for which the `group` dimension is set to `Standard`. Finally, click Apply splitting and choose the Method field. The chart you see now shows the 90th percentile of Server's HTTP response time, and it's grouped by the HTTP method.

The data points you'll see will most likely be different than what's shown in the screenshot above.

 **Note:** If there's no data shown in the dashboard, open the Server dashboard at `<SERVER_URL>/dashboard`, upload a couple documents, and perform some operations on them.

Now you can explore more [PSPDFKit Server metrics](#). When you're satisfied with how the chart looks, add it to the dashboard so that you can get back to it. You can learn more about building advanced charts by following the [Advanced features of Azure Metrics Explorer](#) guide.

---

Was this helpful?

 YES

 NO

---

Questions? [Contact us](#)