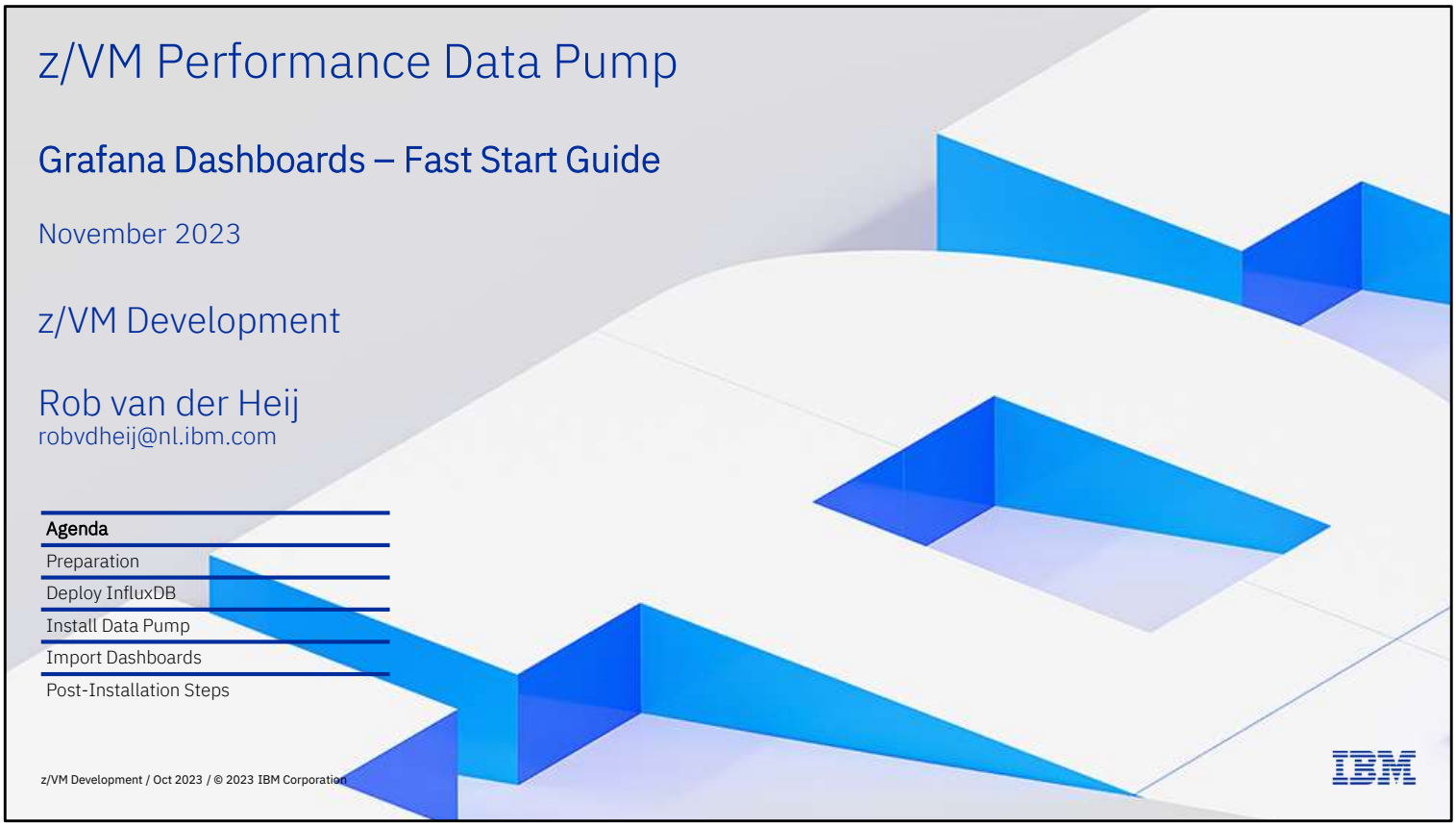


Title: **Grafana Dashboards – Fast Start Guide**

Abstract: This presentation shows a quick way to get your z/VM real-time performance data shown in Grafana dashboards hosted on a Linux guest in your z/VM system. Since this is a solution that bridges the gap between z/VM and popular open-source projects, some of the enablement is done on z/VM, and some is done on Linux. The description is detailed enough to be useful for those with limited experience on one or both sides of the solution, though it helps to have a friendly experienced colleague handy, in addition to an Internet browser. Experienced professionals themselves will be able to fast forward through some of the steps. It is not uncommon to have this all done in a few hours.

Preparation

Preparation is a big part of the work, but it can really help to ensure you have everything you need to get this done. This section is intended as your shopping list to make sure you have all the ingredients in place before you start cooking.

Scan the Presentation

Review the entire presentation to have an overview of the steps that make up the process

The steps are presented in an “optimized workflow” order eliminate excessive context switching when done by yourself

Try to spot things that you must delegate to a colleague and get those things planned

Linux tasks

```
[rob@a3530038 ~]$ find ~/vmprf
/home/rob/vmprf
/home/rob/vmprf/influxdb
/home/rob/vmprf/influxdb/influxdb.conf
/home/rob/vmprf/grafana
/home/rob/vmprf/grafana/grafana.ini
```

z/VM tasks

```
q monitor sample
MONITOR SAMPLE INACTIVE
               INTERVAL    0 MINUTES    PENDING INTERVAL    1 MINUTES
               RATE     STOP            PENDING RATE     2.00 SECONDS
MONITOR DCSS NAME - MONDCSS
CONFIGURATION SIZE     4096 LIMIT         1 MINUTES
CONFIGURATION AREA IS FREE
USERS CONNECTED TO *MONITOR - DATAPUMP           PENDING-CONFIG
MONITOR   DOMAIN ENABLED
SYSTEM    DOMAIN ENABLED
 . . .
SSI       DOMAIN ENABLED
Ready; T=0.01/0.01 15:46:33
```


It helps to go through the presentation once before starting to complete the various steps. That way you know where you need to reach out to a colleague for guidance or even have someone else complete the steps that you’re not supposed to do. Seeing the content also helps to see where you need additional training.

The speed up your scanning, the Linux activities are marked with a red frame around the box, the z/VM system administration tasks are with a blue frame.

Where applicable, the commands are also listed in the notes under the slide to make it easier to copy and paste; be aware that quotes sometimes get beautified such that they don’t work in the shell anymore. When in doubt, edit the command again and replace the quotes.

Order PTF UM90334 for VM66687

Use your normal process to order the service from IBM, using Shopz or Service Link

It may take an hour until the service is available for download; you don't want to be waiting for that

When your normal process is to leave ordering and installing service to a colleague, plan to have that done before you start the remaining steps

Applying the service is non-disruptive; it does not require an IPL, it does not even need to interrupt PERFSVM

The reason for ordering the service early is that it can take some time for the order to ship, so this way you avoid waiting for it later.

When you're not on z/VM 7.3, there is no PTF to order. If you're participating in an early support program for Data Pump, refer to the documentation for that.

Review Security Policy

Note: This description configures InfluxDB and Grafana without authentication and without secure connections. This keeps the setup simple.

This is not sufficient for production environments. The extra effort for a secure setup is almost entirely in the container setup and customization of InfluxDB and Grafana.

DATAPUMP consumes VM performance data through *MONITOR using the MONDCSS segment

Secure TCP/IP connection between DATAPUMP and InfluxDB requires a working z/VM SSL setup

- Linux server needs server certificate
- Root certificate must be in z/VM SSL database
- Grafana requires SSL – this is not related to the Data Pump, but uses same server certificate

Grafana can be configured to use an external LDAP connection for user authentication

When network isolation prevents all z/VM systems sending data to the same InfluxDB service, see whether an exception is acceptable

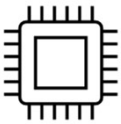
The DATAPUMP virtual machine (to be added) will need special privileges to consume the real-time performance data on z/VM. This requires two special statements in the directory entry for the user. When you want DATAPUMP to also start the z/VM monitor (when you don't have another performance monitor that already does that) you need specific privileges for that.

The Data Pump uses a TCP/IP connection to send the extracted data to the InfluxDB service, whether the Linux guest is running on the same VM system or another one. Your security policy may require that you use secure connections for that, using SSL. While the Data Pump can do that just fine, it requires a working z/VM SSL setup. Since the Data Pump is the "client" on the SSL connection, you don't need an SSL server certificate for this. When you already use secure connections for your TN3270 sessions to z/VM, you already have a working z/VM SSL setup.

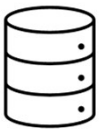
For a secure connection between the Data Pump and InfluxDB, the InfluxDB container requires an SSL server certificate. If that is your company policy, this can be a certificate issued by your internal certificate authority. Whether public or internal certificate, the root certificate to warrant the server certificate must be stored in the z/VM System SSL certificate database.

You will also need an SSL connection for the web browser connecting to Grafana. As both containers run on the same Linux guest, it is convenient to use the same SSL server certificate for both.

Define a Linux Server



On average 1% of CPU, with 5 active users and 10 systems



InfluxDB database as 1 GB per z/VM system, per month of data

Virtual memory around 3 GB depending on the number of systems and resources

You will eventually want the Linux server in a production environment

Linux server uses only minimal resources

Ensure Linux can be reached from the z/VM systems where Data Pump will be running

The following description is based on RHEL 9.2 with podman installed

Install **podman-docker** to use Docker commands

Rootless containers are owned by a non-root user

- You may need a functional user on Linux
- Persistent storage volumes in home directory

As the InfluxDB service gathers data from all your z/VM systems, you will eventually run this in the most reliable environment. There's nothing wrong trying it first in a test configuration.

The Linux server takes only minimal resources. Rule of thumb is about 1% of CPU to monitor 10 z/VM systems with some Linux guests, and maybe 5 active users working with dashboards. The disk space required depends on the number of systems that you monitor, and on how long you want to retain the detailed data. Prepare for about 1 GB of disk space per z/VM system; ideally using LVM so you can add space when necessary. A virtual machine with 3 GB of memory is likely enough for 10 z/VM systems and lets you run a second set of containers for testing.

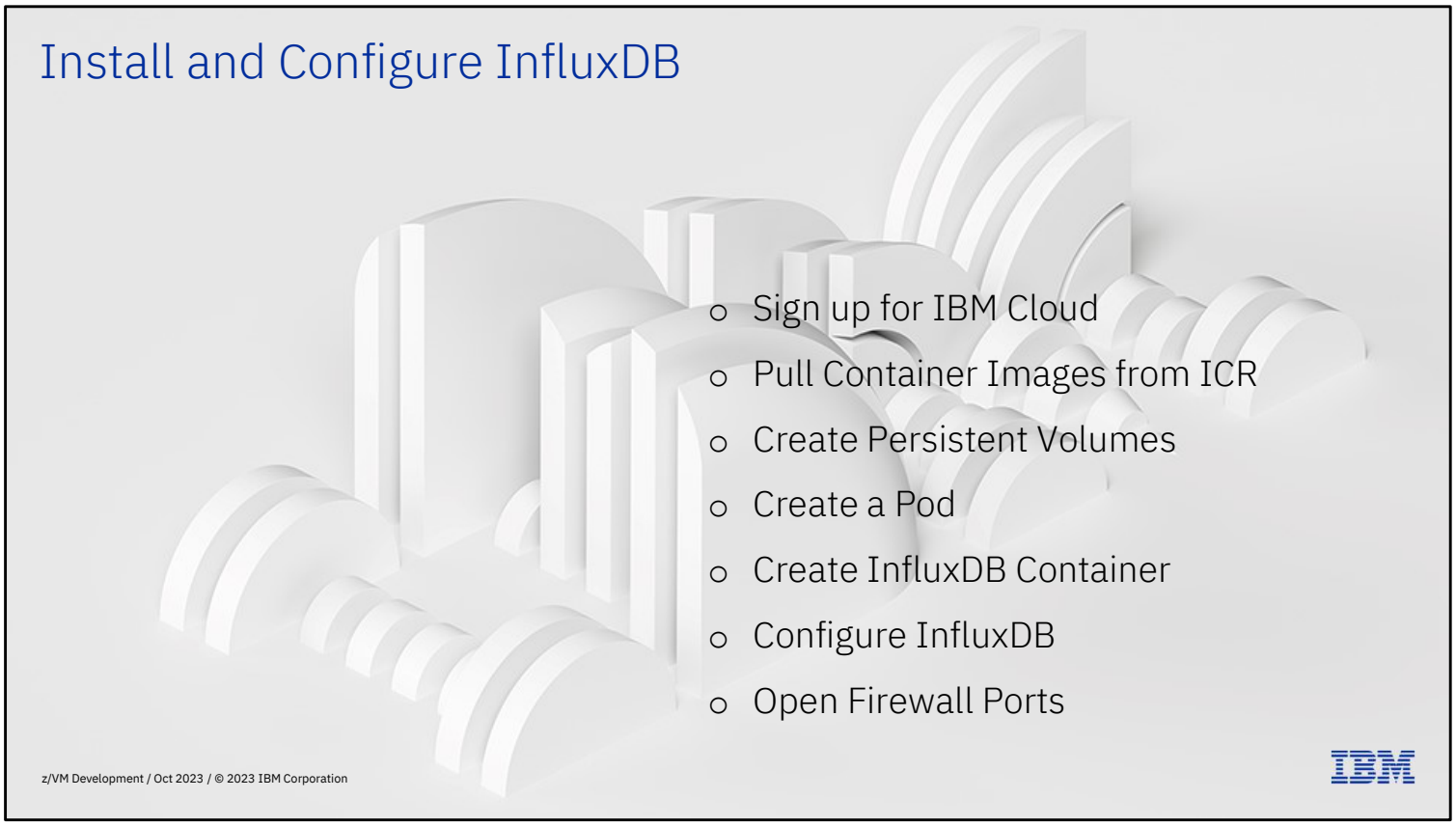
The z/VM systems that you monitor with the Data Pump must be able to connect to the InfluxDB service in the Linux server. You may need firewall rules to allow such connections. If you really need to keep systems isolated, you may have to replicate the Linux setup in different subnets.

This description is based on a RHEL 9.2 system with **podman** installed. Most things work similar when you are using Docker. If you have podman but are used to Docker commands, there's a Red Hat package with a **docker** emulation.

The rootless containers are owned by a non-root user. You may have requirements to define a functional user for that, and not use your own personal Linux account. When you need to mount disk space to hold the persistent data and the podman images, be aware that these files are created in the home directory of the owner.

Note: As InfluxDB v1 and Grafana are not cloud-native applications, there is no benefit in running the containers

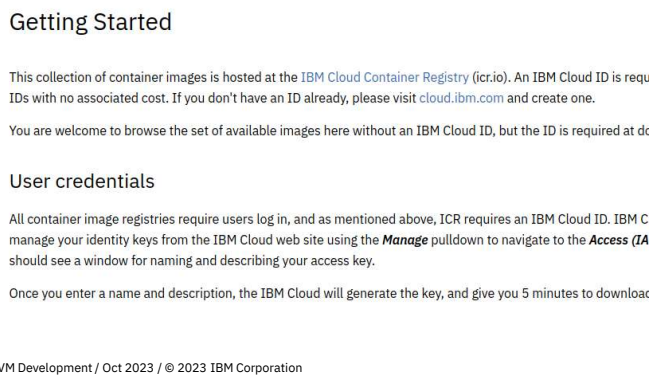
under control of OpenShift or Kubernetes. It is probably simpler to add a small Linux guest just for this.



We start with the InfluxDB container. To install InfluxDB, you need to get the container images and create the persistent volumes and the pod. We then create the InfluxDB container and configure it.

The reason to do this first is to have a working InfluxDB service when working on the Data Pump configuration, so there is a place to send the data and see that things are happening.

Sign up for IBM Cloud



No associated cost with IBM Z Container Registry

Follow instructions on IBM Z Container Registry

Sign up for an IBM Cloud ID if necessary

Create IBM Cloud API key

Download the JSON document with credentials

Keep it handy on the Linux guest


```
{
    "name": "icr.io key",
    "description": "IBM Z Container Registry credentials",
    "createdAt": "2023-08-08T11:49+0000",
    "apikey": "_0X-1wvKblAYLh2TqLlTrtXGsKeZ6XhZ5JE3isr2G5HP"
}
```

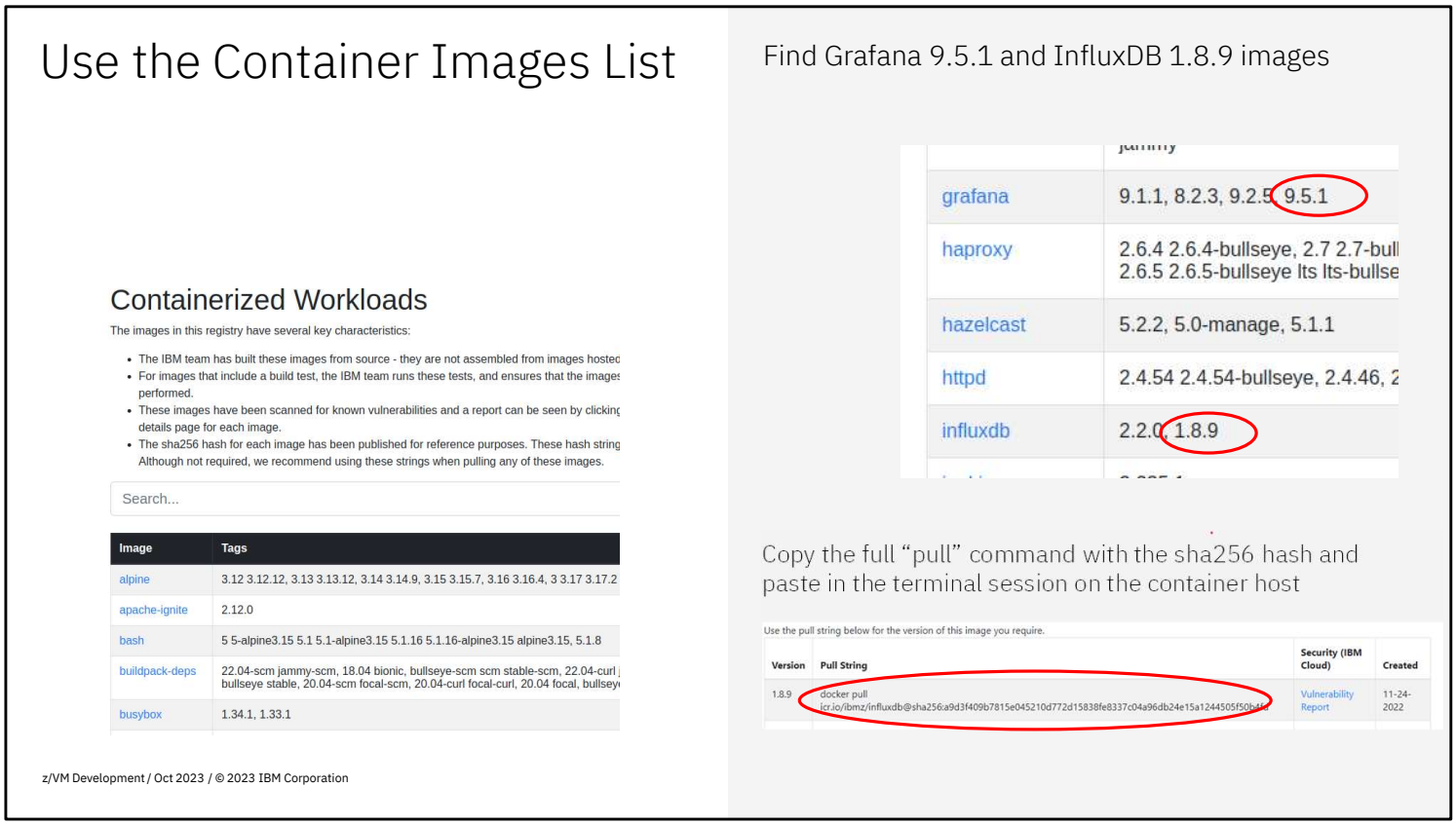

https://ibm.github.io/ibm-z-oss-hub/main/main.html

Visit the IBM Z Container Registry site for the instructions to get access. You will need an IBM Cloud ID and create an API key to access the repository. When you create the API key, you can copy the key itself to the desktop notepad or download the JSON document. I prefer to take the JSON document and store it somewhere on the Linux server. Since it has credentials, I find it handy to store it in the .ssh directory.

The “apikey” value is what you normally need as authentication.

Note: The example shown here does not contain valid credentials; you must obtain your own credentials.

Use the Container Images List



Once you accepted the IBM Open Source Container Image Agreement, you can view the list of container images. For this project, we need the Grafana 9.5.1 image and the InfluxDB 1.8.9 image.

Don’t be tempted to pick the InfluxDB v2 unless you must. It does not offer many advantages and is harder to configure for Data Pump. We expect that v3 will be available early in 2024 to solve the major issues with v2.

Pull IBM Z Container Images

Use the API key created in IBM Cloud

```
[rob@a3530037 ~]$ podman login -u iamapikey -p `jq -r '.apikey' < ~/.ssh/icr.io.apikey` icr.io
Login Succeeded!
[rob@a3530037 ~]$ docker pull icr.io/ibmz/influxdb@sha256:a9d3f409b7815e04...
...
Writing manifest to image destination
Storing signatures
33835fc028733def11186f59e90d678922a7e3af782b29e0493e9a6ab372ae8e
[rob@a3530037 ~]$ docker pull icr.io/ibmz/grafana@sha256:0c46580dc8837f727...
...
Writing manifest to image destination
Storing signatures
51757b50300774d9c61b60a99fe98016de0c980096e95018ae22f91e8109364b

[rob@a3530037 ~]$ podman image ls
REPOSITORY            TAG         IMAGE ID      CREATED       SIZE
icr.io/ibmz/grafana   <none>      51757b503007  2 months ago  450 MB
icr.io/ibmz/influxdb  <none>      33835fc02873  7 months ago  257 MB
[rob@a3530037 ~]$ podman tag 51757b503007 icr.io/ibmz/grafana:9.5.1
[rob@a3530037 ~]$ podman tag 33835fc02873 icr.io/ibmz/influxdb:1.8.9
```


The **podman login** uses the **jq** command to retrieve the key from the JSON document. If you don't have the **jq** package installed and don't want to do that, you can just copy and paste the value from the document into the command line.

After login, you can **pull** (download) the container images from the registry. I find it practical to **tag** the images with the version number, so it is easier to tell what you have.

```
podman login -u iamapikey -p `jq -r '.apikey' < ~/.ssh/icr.io.apikey` icr.io
docker pull icr.io/ibmz/influxdb@sha256:a9d3f409b7815e04...
docker pull icr.io/ibmz/grafana@sha256:0c46580dc8837f727...
podman tag 51757b503007 icr.io/ibmz/grafana:9.5.1
podman tag 33835fc02873 icr.io/ibmz/influxdb:1.8.9
```

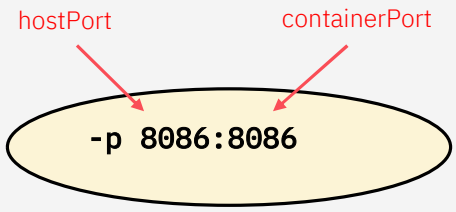
Create Persistent Volumes and Pod

Create volumes for InfluxDB and Grafana

```
[rob@a3530038 ~]$ podman volume create influxdb
influxdb
[rob@a3530038 ~]$ podman volume create grafana
grafana
```

Create a pod for the containers

```
[rob@a3530038 ~]$ podman pod create --name vmprf -p 8086:8086 -p 3000:3000
8c7c2db87dd8bc033aca0460aa89d61213e78b85fc621b3473f643ab3c6d18a1
[rob@a3530038 ~]$ podman pod ls
POD ID        NAME        STATUS      CREATED         INFRA ID      # OF CONTAINERS
8c7c2db87dd8  vmprf       Created     37 seconds ago  54ed30fa043a  1
```



You define persistent storage for the InfluxDB database and the Grafana configuration database, so the data is preserved when the container is stopped and created from scratch, for example for an upgrade.

Define a pod for the two containers. Because InfluxDB and Grafana need a TCP/IP connection to the outside, we define those ports when the pod is created. In this case we use the same port number inside and outside the pod, but if you would run a second pod for testing on the same host, this is where you specify the different outside port numbers.

Note: When using Docker, there is no “pod” concept. You simply define the InfluxDB and Grafana containers as separate objects. You may find docker-compose convenient to bundle the container setup.

```
podman volume create influxdb
podman volume create grafana
podman pod create --name vmprf -p 8086:8086 -p 3000:3000
```

Create InfluxDB Configuration

Store data where we mount the volume in the container

Prepare a configuration file

```
[rob@a3530038 ~]$ mkdir -p ~/vmprf/influxdb
[rob@a3530038 ~]$ cat > ~/vmprf/influxdb/influxdb.conf << EOL
reporting-disabled = true
bind-address = ""
[meta]
  dir = "/var/lib/influxdb/meta"
[data]
  dir = "/var/lib/influxdb/data"
  wal-dir = "/var/lib/influxdb/wal“
[http]
  auth-enabled = true
EOL
```


We need to create a small configuration file for InfluxDB since the defaults from the module don't work out of the box. Since we need a few more things there later, I like to use a subdirectory named after the pod itself, so it's easy to find things. You can use whatever naming convention you consider wise. The name influxdb.conf is what the database expects to see.

The configuration file specifies that the database is stored in /var/lib/influxdb/ which is still inside the container. You see later where that data ends up in the host. The bind-address option is some old setting that appears still set by default and prevents the InfluxDB container from running. And finally, there's a reporting-disabled option that you can set to true to prevent a call-home to InfluxData; comply with your company policy when setting this value.

The last section enables authentication for InfluxDB.

```
mkdir -p ~/vmprf/influxdb
cat > ~/vmprf/influxdb/influxdb.conf << EOL
reporting-disabled = true
bind-address = ""
[meta]
  dir = "/var/lib/influxdb/meta"
[data]
  dir = "/var/lib/influxdb/data"
  wal-dir = "/var/lib/influxdb/wal“
```

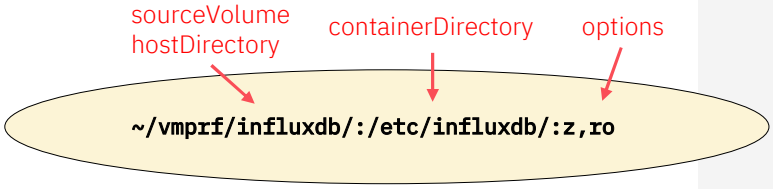
```
[http]
  auth-enabled = true
EOL
```

Create InfluxDB Container

Add the InfluxDB container to the pod

Set environment variables to create admin user rather than admin/admin

```
[rob@a3530038 ~]$ podman create --pod vmprf --name influxdb \
> -e INFLUXDB_ADMIN_USER=rob \
> -e INFLUXDB_ADMIN_PASSWORD=verysecret \
> -v ~/vmprf/influxdb/:/etc/influxdb/:z,ro \
> -v influxdb:/var/lib/influxdb \
> icr.io/ibmz/influxdb:1.8.9
86ecd9d2a7cfa21a1f6e1e64f0017b124e01b9a6e2081eb1d9e31f52adfa2198
```




```
[rob@a3530038 ~]$ podman pod ps
POD ID        NAME        STATUS      CREATED             INFRA ID      # OF CONTAINERS
a8900232bd0a  vmprf       Created     About a minute ago  dddc77c34a8c  2
```


The **podman create** looks a bit intimidating because of all the options. The first line references the pod that was created before, and names this container '**influxdb**'.

```
podman create --pod vmprf --name influxdb \
            -e INFLUXDB_ADMIN_USER=rob \
            -e INFLUXDB_ADMIN_PASSWORD=verysecret \
            -v ~/vmprf/influxdb/:/etc/influxdb/:z,ro \
            -v influxdb:/var/lib/influxdb \
            icr.io/ibmz/influxdb:1.8.9
```

The two **-e** options set the environment variables such that a unique admin user is created with specified password, rather than the default admin/admin. Pick your own name and password.

Following are two options to mount volumes into the container. The first volume statement mounts the subdirectory we just created into the container at **/etc/influxdb/** which means that the application finds that configuration file. The **"z"** option is for se-linux to let the container access the host files; you probably expected the **"ro"** to mean that it is mounted Read/Only so the container does not change the configuration file. The second volume statement mounts the persistent volume named "influxdb" (that was created a few slides earlier) in the container at /var/lib/influxdb/ which is exactly what we put in the configuration file.

The **podman pod ps** can be used to show that we have a pod with two containers (one being a dummy just to hold

the pod). The pod is still stopped, since we just created it.

InfluxDB Administration

Start the Pod

```
[rob@a3530038 ~]$ podman pod start vmprf
86901d8ec290302f68186adbe7e26c768f18cfdb01609b3e501b77fc2bb13380
[rob@a3530038 ~]$ podman pod ps
POD ID        NAME        STATUS      CREATED     INFRA ID      # OF CONTAINERS
86901d8ec290  vmprf       Running     2 days ago  cd95edbb0052  2
```

Check the InfluxDB logging

```
[rob@a3530038 ~]$ podman logs influxdb
ts=2023-08-16T10:22:59.651801Z lvl=info msg="InfluxDB starting" log_id=0jgTlUGl000 version=1.8.9 branch=HEAD commit=d9b56321d5796d77911
ts=2023-08-16T10:22:59.651884Z lvl=info msg="Go runtime" log_id=0jgTlUGl000 version=go1.17.8 maxprocs=2
ts=2023-08-16T10:22:59.653276Z lvl=info msg="Using data dir" log_id=0jgTlUGl000 service=store path=/var/lib/influxdb/data

ts=2023-08-16T10:23:00.979258Z lvl=info msg="Listening on HTTP" log_id=0jgTlUGl000 service=httpd addr=[::]:8086 https=false
ts=2023-08-16T10:23:00.979311Z lvl=info msg="Starting retention policy enforcement service" log_id=0jgTlUGl000 service=retention
check_interval=30m
ts=2023-08-16T10:23:00.979547Z lvl=info msg="Listening for signals" log_id=0jgTlUGl000
```

Define the database for Data Pump

```
[rob@a3530038 ~]$ podman exec -it influxdb influx  -username rob -password verysecret \
> -execute "create database zvm;alter retention policy autogen on zvm duration 4w; show databases"


name: databases
name
----
_internal
zvm
```


We now have a stopped pod with InfluxDB, so we can use a **podman pod start** command to get that running. The **podman pod ps** shows that it worked.

You might still want to use **podman logs influxdb** so you know what a good startup looks like. When there are obvious errors, this is where you start looking for mistakes.

We now need to issue **influx** commands inside the container to configure a new database. Since we already enabled authentication, you need to provide the admin username and password.

An easy way is to use the **podman exec** command which runs the **influx** user interface to issue a single command. Alternatively, you can just run the **influx** user interface without the **–execute** option and issue commands interactively.

```
podman pod start vmprf
podman exec -it influxdb influx  -username rob -password “verysecret” \
           -execute “create database zvm; alter retention policy autogen on zvm
duration 4w; show databases”
```

The commands shown above crate the “zvm” database and set a retention period of 4 weeks. Adjust now (or later) when you think you need a different retention of the data.

Bonus: The Influx Client

Copy the **influx** binary from the container

```
[rob@a3530038 ~]$ mkdir -p ~/bin
[rob@a3530038 ~]$ podman cp influxdb:/usr/bin/influx ~/bin/
[rob@a3530038 ~]$ file ~/bin/influx
/home/rob/bin/influx: ELF 64-bit MSB executable, IBM S/390, version 1 (SYSV), dynamically linked, interpreter
[rob@a3530038 ~]$ influx -version
InfluxDB shell version: 1.8.9
```

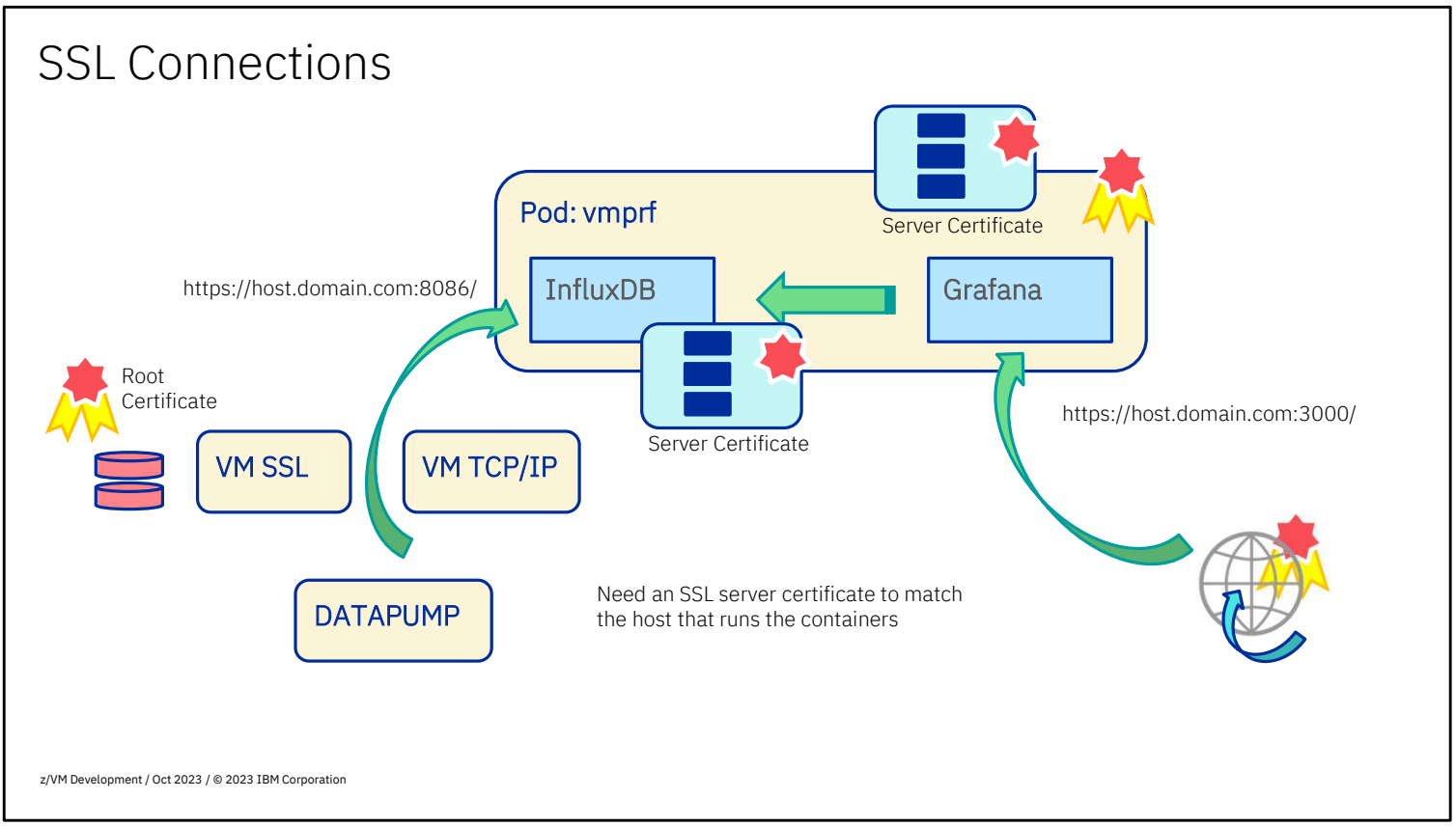
```
[rob@a3530038 ~]$ influx -username rob -password verysecret -execute 'show databases'
name: databases
name
----
_internal
zvm
```

```
[rob@a3530038 ~]$ influx -username rob -password verysecret
Connected to http://localhost:8086 version 1.8.9
InfluxDB shell version: 1.8.9
> use _internal
Using database _internal
> select * from "write" where time > now() - 1m
name: write
time                hostname pointReq pointReqLocal req   subWriteDrop subWriteOk writeDrop writeError writeOk writeTimeout
----                -------- -------- ------------- ---   ------------ ---------- --------- ---------- ------- ------------
1692268390000000000 vmprf    1980068  1980068       16050 0            16050      0         0          16050   0
1692268400000000000 vmprf    1980106  1980106       16051 0            16051      0         0          16051   0
1692268410000000000 vmprf    1980144  1980144       16052 0            16052      0         0          16052   0
1692268420000000000 vmprf    1980182  1980182       16053 0            16053      0         0          16053   0
1692268430000000000 vmprf    1980220  1980220       16054 0            16054      0         0          16054   0
1692268440000000000 vmprf    1981647  1981647       16061 0            16061      0         0          16061   0
```


Using the **docker exec** command to work with InfluxDB is nice, but it gets tedious when doing more serious work with InfluxDB. There is an elegant trick to directly use the **influx** client program that we also have in the container. The first step is to copy the program from the container to your userid on the same system.

The default for the **influx** client is to use **localhost** and port **8086** which means that the client talks directly to the service running inside the container. You must provide the admin username and password on the command. You can also store those in environment variables INFLUX_USERNAME and INFLUX_PASSWORD to avoid typing it each time. You can still use this with the **-execute** option and avoid the **docker exec** around it, but you can also use the interactive InfluxDB shell as shown.

You can even use the InfluxDB shell on your workstation, but the s390x executable probably wouldn't do. Instead, you would get a suitable container image for your workstation and borrow the influx program from it.



Note: You can create a working solution without authentication and without SSL connections, but it probably does not comply with your security policy. Understand the implications when you skip the following two slides.

To establish an SSL connection, both InfluxDB and Grafana services need needs an SSL server certificate. Because both containers run on the same host, a single certificate work for both. Depending on your company policy, such a server certificate is either purchased from a well-known commercial root authority, or an internal organization produces the certificates. With the server certificate comes the root certificate that warrants authenticity.

For DATAPUMP to establish an SSL connection to InfluxDB, the root certificate is stored in the VM SSL certificate database. The server certificate is installed in the InfluxDB container instance.

For the web browser on the user's workstation to connect securely with Grafana, the Grafana container also needs that server certificate. When the server certificate was signed by one of the commercial well-known authorities, the browser already has the corresponding root certificate. For company managed certificates, your workstation probably already has the root certificate installed as well.

The Grafana service also needs to connect to InfluxDB to request data from the database. As that happens inside the pod, we could argue that it doesn't need to use SSL. Since Data Pump and Grafana use the same REST API, so it's all or nothing. This means that Grafana is both an SSL server (for the browser connection) and an SSL client (for the InfluxDB connection). The Grafana container gets both the signed certificate for the SSL server role, and the root certificate for the SSL client role.

When you generate a Certificate Signing Request (CSR) for your server, include the localhost and 127.0.0.1 as additional address. It is also helpful to include the IP address of the Linux server in the certificate request, since that makes it possible for Data Pump to bypass the DNS server if needed.

Hardening InfluxDB

Create a user with password for each z/VM system

```
[rob@lnxrmh01 ~]$ docker exec -it test-influxdb /bin/influx -execute "create user datapump with password 'Secret123'"
[rob@lnxrmh01 ~]$ docker exec -it test-influxdb /bin/influx -execute "grant write on zvm to datapump"

[rob@lnxrmh01 ~]$ docker exec -it test-influxdb /bin/influx -execute "create user grafana with password 'Just2read'"
[rob@lnxrmh01 ~]$ docker exec -it test-influxdb /bin/influx -execute "grant read on zvm to grafana"
[rob@lnxrmh01 ~]$ docker exec -it test-influxdb /bin/influx -execute "grant read on _internal to grafana"
```

Enable authentication and SSL

```
[rob@lnxrmh01 influxdb]$ cat >> ./influxdb.conf << EOL
[http]
  auth-enabled = true
  https-enabled = true
  https-certificate = "/etc/influxdb/lnxrmh01.pem"
  https-private-key = "/etc/influxdb/lnxrmh01.key"
EOL
```

```
[rob@lnxrmh01 ~]$ tree ~/vmprf/*
/home/rob/vmprf/influxdb
├── influxdb.conf
├── lnxrmh01.key
└── lnxrmh01.pem
```

Verify after restarting the container

```
[rob@lnxrmh01 ~]$ podman pod restart vmprf
486fa3654503923483 9ef64415237b2b4369ad82c7fc29c5bb52d8cbafb12756
[rob@lnxrmh01 ~]$ curl https://lnxrmh01.pok.ibm.com:8086/ping?verbose=1
{"version":"1.8.9"}
```

https://docs.influxdata.com/influxdb/v1.8/administration/authentication_and_authorization/

Now that we have the mechanics of the container working, we need to add authentication and enable security options. Even though we created that administrator with secret password, user authentication is not enabled yet. That is convenient while doing the setup. There is little risk because the Linux firewall prevents external access to the InfluxDB service.

We define the user **datapump** in InfluxDB to allow the DATAPUMP virtual machine to write data into the **zvm** database. When you have multiple z/VM systems feeding data, you may want to define the user to match the system name of the z/VM system, so you can have unique credentials for each of them.

Also define a user for the Grafana dashboards; unlike the ones for Data Pump, this one needs to **read** the databases. The **_internal** database is used by InfluxDB itself to provide metrics on database usage, which can be helpful to monitor InfluxDB.

For SSL connections, the InfluxDB service needs a server certificate. You will also need that for Grafana, and with the two containers on the same host, you can use the same server certificate. Since you're using the same certificates for both servers, it makes sense to name the certificate and key after the server rather than the service.

The **curl** command shows that we can make a proper SSL connection to the container. If you want to take it a step further, experiment with **curl** commands to show the databases or issue other InfluxQL queries.

Open Firewall Ports

Open the firewall ports for InfluxDB

```
[rvdheij@a3530038 ~]$ sudo firewall-cmd --add-port=8086/tcp
success
[rvdheij@a3530038 ~]$ sudo firewall-cmd --add-port=8086/tcp --permanent
success
```

Allow containers to run after logging off

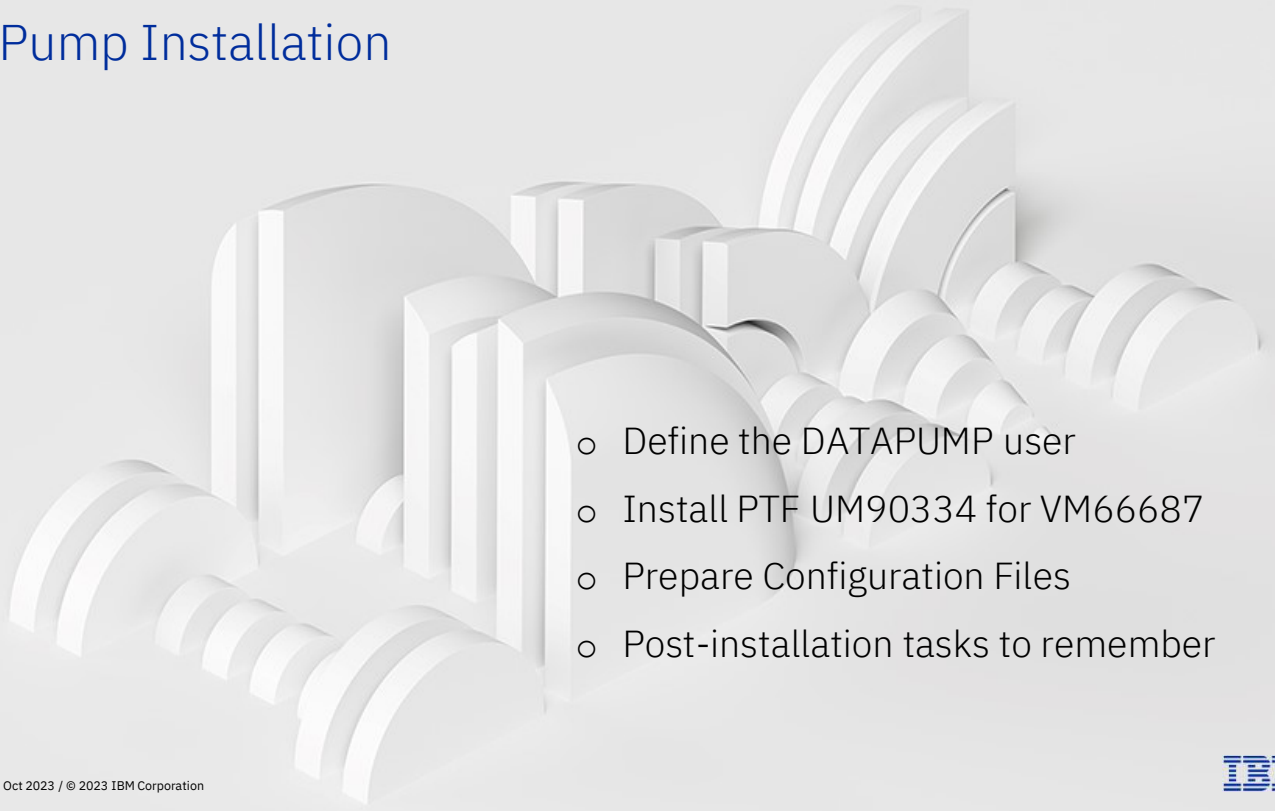
```
[rvdheij@a3530038 ~]$ sudo loginctl enable-linger rob
```


For the Data Pump in z/VM to send data, we also need to configure the firewall to let a connection to port 8086 through. We issue the command twice, first for the currently active configuration, and next with the –permanent option to have it defined after a reboot. If you want to restrict traffic to just the z/VM system that you expect to send data with Data Pump, use the appropriate extra parameters to restrict connections further.

Before we forget, the rootless container in Podman require a loginctl command to let the container run after you logoff.

You now have a running InfluxDB accepting connections from the Data Pump. It’s time to install the Data Pump code.

Data Pump Installation



- Define the DATAPUMP user
- Install PTF UM90334 for VM66687
- Prepare Configuration Files
- Post-installation tasks to remember

Define the DATAPUMP user

```
IDENTITY DATAPUMP LBYONLY 128M 512M EG
 INCLUDE IBMDFLT
 ACCOUNT IBM
 NAMESAVE MONDCSS
 IUCV *MONITOR MSGLIMIT 255
 SHARE ABS 1%
 IPL CMS PARM AUTOCR FILEPOOL VMSYS:
 OPTION SVM
 LOGONBY IBMVM1
 LINK PERFSVM 201 201 RR
```

Refer to the “z/VM: Performance” publication, Appendix I for details.

Privilege class E is only needed when DATAPUMP should start the z/VM Monitor

Data Pump code will be installed on PERFSVM 201

Define profiles and permissions in your ESM

Different ways to prepare the SFS setup:

- Enroll by hand and create a PROFILE EXEC
- Wait until service is applied and use MDXSETUP

https://www.ibm.com/docs/en/zvm/7.3?topic=pump-setting-up-datapump-virtual-service-machine

The “z/VM Performance” publication has a new Appendix I on the Data Pump that documents these steps.

The DATAPUMP user is created like the directory entry shown. If you use DIRMAINT, you create the file and issue a **DIRM ADD DATAPUMP** to create the user. A few things to notice here:

You normally don’t need to logon to the DATAPUMP virtual machine but leave it running unattended. Initially, you may need to fix things and might want to logon. You could later change it to **AUTOONLY** and just have it automatically started after an IPL.

The privilege class **E** as shown is for when you need DATAPUMP to start the monitor. If you keep PERFSVM running as well, then PERFSVM takes already care of starting the monitor.

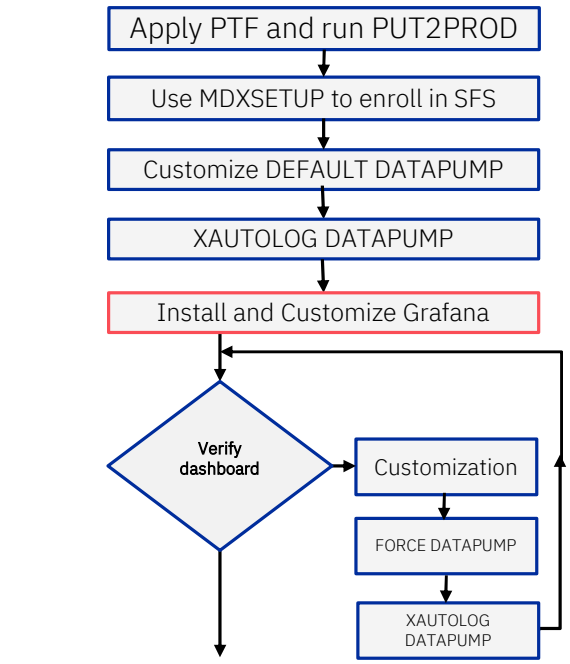
The **NAMESAVE** and **IUCV** statements as shown are typical for a virtual machine that consumes monitor data (you will see them in PERFSVM as well).

A **SHARE ABS 1%** is to ensure that DATAPUMP gets the resources to collect performance data even when the system is busy; that’s when you want to gather data, so you don’t want the performance monitor getting left behind.

The **IPL CMS** specifies **FILEPOOL VMSYS:** to get the PROFILE EXEC and configuration files from Shared File System. This has the advantage that you don’t need to allocate a minidisk for DATAPUMP on each member of the SSI.

The disadvantage of using VMSYS: for the A-disk is that you need to enroll DATAPUMP on each member and need to maintain the configuration files on each member. You can do that all by hand, but we also provide a utility to do that for you. If you are in a hurry and don’t expect to have very special requirements, then it might make sense to use the MDXSETUP utility for that.

Apply PTF and Configure Data Pump



Download the service with GETSHOPZ and apply the service using MAINT730

Suggested order:

1. Apply PTF UM90334
2. PUT2PROD on first member
3. use MDXSETUP to enroll in VMPSFS: and VMSYS:
4. Complete customization to make DATAPUMP work
5. Install and customize Grafana and import a dashboard
6. Verify that Grafana dashboard is working
7. Post-Installation Steps

If you have an SFS filepool shared even wider than the VMPSFS: file pool for z/VM service (for example through IPGATE or AVS) then you can use that to hold the primary copy of the customization files. Specify the file pool name as argument for the MDXSETUP utility.

To avoid double work, first configure Data Pump on a single z/VM system (the one where you're logged on to apply the service and run PUT2PROD). Run MDXSETUP to enroll DATAPUMP in the file pools, and to prepare the directories.

Note: When you're on z/VM 7.2 and using the material for the early support program, follow those instructions instead of applying the PTF and running PUT2PROD. Use MAINT720 in that case.

Run PUT2PROD

```
 . . .
VMFP2P1231I Copying files from DIR VMPSFS:7VMPTK30.PERFTK.SAMPLE to PERFSVM 1CC
VMFP2P2204I Linking PERFSVM 201 as 1FFF with link mode M
VMFP2P1231I Copying files from DIR VMPSFS:7VMPTK30.PERFTK.TBUILD to PERFSVM 201
VMFP2P1233I The following products have been put into production. Recycle the appropriate servers.
VMFP2P1233I PERFTKSFS
VMFP2P2264I Restoring prior system environment using saved access/minidisk information
VMFSET2760I VMFSETUP processing started for ENVRESTORE PUT2PRODEXEC20230810175752
VMFSET2760I VMFSETUP processing completed successfully (RC=0)
VMFP2P2760I PUT2PROD processing completed successfully (RC=0)
Ready; T=55.41/61.28 17:59:34
```

Run MDXSETUP

```
vmlink PERFSVM 201 PERFSVM 201 PERFSVM 1CC ( invoke mdxsetup <filepool>
DMSVML2060I PERFSVM 201 linked as 0120 file mode X
DMSVML2060I PERFSVM 1CC linked as 0121 file mode W
Updating files:
DEFAULT  DATAPUMP  2023-08-10 18:07:38 VMPSFS:DATAPUMP.
PROFILE  EXEC      2023-08-10 18:07:38 VMPSFS:DATAPUMP.
Customize configuration files with VMLINK .DIR VMPSFS:DATAPUMP. ( FILEL
DMSVML2061I PERFSVM 201 detached
Ready; T=0.04/0.06 18:07:38
```


The MDXSETUP utility can be used to enroll DATAPUMP in the SFS file pools and prime the A-disk with a PROFILE EXEC and configuration file. When you have multiple shared file pools, specify the name of the shared file pool on the MDXSETUP command.

Customize Data Pump

```
 MAINT730 FILELIST A0  V 169  Tru
Directory = VMPSFS:DATAPUMP.
Cmd   Filename Filetype Fm Format
      DEFAULT  DATAPUMP Z1 V
      PROFILE  EXEC     Z1 V
      VMA               Z  DIR
```

Use the VMLINK command as suggested

```
VMLINK .DIR VMPSFS:DATAPUMP. ( FILEL
```

Add the InfluxDB section to DEFAULT DATAPUMP

only when no other service starts the z/VM monitor

```
[monitor]
  type=monitor
  start = true

[influxdb]
  type = influxdb
  url = https://datapump:Secret123@linux007.ibm.com:8086/
```

URL of your InfluxDB service

Review PROFILE EXEC for any extra requirements

When you use the VMLINK command as suggested in the previous slide, you will see the directory with two files. Edit the DEFAULT DATAPUMP and add the InfluxDB section. If you have no other service that would start the monitor, add the extra line for the monitor section as well.

The URL for InfluxDB should contain the user and password that you defined earlier when configuring InfluxDB.

Checking after XAUTOLOG DATAPUMP

- See that DATAPUMP has connected to *MONITOR
- When INACTIVE something needs to start the monitor

  `start = true`

```
q monitor sample
MONITOR SAMPLE INACTIVE
               INTERVAL    0 MINUTES    PENDING INTERVAL    1 MINUTES
               RATE     STOP            PENDING RATE     2.00 SECONDS
MONITOR DCSS NAME - MONDCSS
CONFIGURATION SIZE     4096 LIMIT         1 MINUTES
CONFIGURATION AREA IS FREE
USERS CONNECTED TO *MONITOR - DATAPUMP           PENDING-CONFIG
MONITOR   DOMAIN ENABLED
SYSTEM    DOMAIN ENABLED
PROCESSOR DOMAIN DISABLED
STORAGE   DOMAIN DISABLED
USER      DOMAIN DISABLED
I/O       DOMAIN DISABLED
NETWORK   DOMAIN DISABLED
ISFC      DOMAIN DISABLED
APPLDATA  DOMAIN DISABLED
SSI       DOMAIN DISABLED
Ready; T=0.01/0.01 13:32:48
```


When you start DATAPUMP, the “query monitor sample” command will show you that DATPUMP has connected (it will take up to two minutes to get two samples and start sending data).

Extra Credits: Check InfluxDB

Use Influx CLI to check for the data

```
[rob@lnxrmh01 ~]$ influx -database zvm -username rob -password verysecret \
                       -hostname lnxrmh01.pok.ibm.com -ssl -execute 'show measurements'
name: measurements
name
----
datapump
iodvsw
mtrisc
mtrmem
```

```
[rob@lnxrmh01 ~]$ influx -database zvm -username rob -password verysecret \
                       -hostname lnxrmh01.pok.ibm.com -ssl -execute 'select * from datapump'
name: datapump
time                elapsed start                system   userid
----                ------- -----                ------   ------
1692195668630000000 0       16 Aug 2023 16:21:08 BOEA3530 DATAPUMP
```


When you want to take it a step further, use the **docker exec** command again to issue **influx** commands in the container. When **show measurements** displays a list of measurements, that's a clear indication that the Data Pump is working. It may take a few minutes to prime the pump. The **datapump** measurement is likely the first to show. You can also display the metrics in that measurement.

Install and Customize Grafana

- Create Grafana Container
- Customize Grafana
- Define InfluxDB Data Source

Now that InfluxDB is available and DATAPUMP is sending data, it is time to install and configure Grafana for the dashboards.

Create Grafana Container

Prepare a configuration file

```
[rob@a3530038 ~]$ mkdir -p ~/vmprf/grafana
[rob@a3530038 ~]$ touch ~/vmprf/grafana/grafana.ini
```

Add the Grafana container to the pod

```
[rob@a3530038 ~]$ podman create --pod vmprf --name grafana \
                      -v ~/vmprf/grafana/:/etc/grafana/:z,ro \
                      -v grafana:/var/lib/grafana \
                      -e GF_SECURITY_ADMIN_USER='rob' \
                      -e GF_SECURITY_ADMIN_PASSWORD='verysecret' \
                      icr.io/ibmz/grafana:9.5.1
41a63b2ed85f63703a423dab93d799e33631a70e303f5ad65bb1618e0d9d14c4
```

```
[rob@a3530038 ~]$ podman ps -a
CONTAINER ID  IMAGE                                  COMMAND     CREATED        STATUS          . .
cd95edbb0052  localhost/podman-pause:4.0.2-1648830545             2 days ago     Up 4 hours ago  . .
21cf4b206528  icr.io/ibmz/influxdb:1.8.9             influxd     4 hours ago    Up 4 hours ago  . .
41a63b2ed85f  icr.io/ibmz/grafana:9.5.1                          7 seconds ago  Created         . .
[rob@a3530038 ~]$ podman start grafana
grafana
[rob@lnxrmh01 ~]$ curl http://lnxrmh01.pok.ibm.com:3000/
<a href="/login">Found</a>.
```


Much of the work for Grafana was already done with InfluxDB, and some is very similar to what was done for InfluxDB. The empty configuration file is to make sure you have a place to put it and have a configuration directory with future changes.

The container is added to the pod very much like we did with InfluxDB. The persistent volume for Grafana is for configuration data und usage statistics that should be carried forward to a new container when you upgrade the code.

The special part is with the **–e** option where we define the Grafana **admin** user with password. You should pick your own user and password. If you're security minded, you may also want to change that again once you are logged on as administrator, and effectively use this as a one-time-password. The default is to use **admin/admin** but since we need to open the firewall to connect with the browser, you shouldn't use such defaults. But if you do, you will be prompted by Grafana after login to change the password.

Hardening Grafana

Enable SSL and add certificates

```
[rob@lnxrmh01 ~]$ cat >> ~/vmprf/grafana/grafana.ini << EOL
[server]
  protocol = https
  cert_file = /etc/grafana/lnxrmh01.pem
  cert_key = /etc/grafana/lnxrmh01.key
EOL

[rob@lnxrmh01 ~]$ tree ~/vmprf
/home/rob/vmprf
├── grafana
│   ├── grafana.ini
│   ├── lnxrmh01.key
│   └── lnxrmh01.pem
└── influxdb
    ├── influxdb.conf
    ├── lnxrmh01.key
    └── lnxrmh01.pem
```

Restart pod and open firewall

```
[rob@lnxrmh01 ~]$ podman pod restart vmprf
41709510250f72d700e7eb900bf2db5ed04ad580dd7ce20af63848b8deb34672
```

```
[rvdheij@a3530038 ~]$ sudo firewall-cmd --add-port=3000/tcp
success
[rvdheij@a3530038 ~]$ sudo firewall-cmd --add-port=3000/tcp --permanent
success
```


For Grafana we also need to enable SSL - authentication is enabled by default already in Grafana. We use the same certificates as for InfluxDB.

Bonus: Create YAML File

Use generate kube to produce a yaml file

```
[rob@lnxrmh01 ~]$ podman pod ps
POD ID        NAME        STATUS      CREATED      INFRA ID      # OF CONTAINERS
41709510250f  vmprf       Running     2 hours ago  7ba683a3554f  3

[rob@lnxrmh01 ~]$ podman generate kube vmprf -f vmprf.yaml
```

The warning messages about truncation of the long hash values don't seem to affect the function.

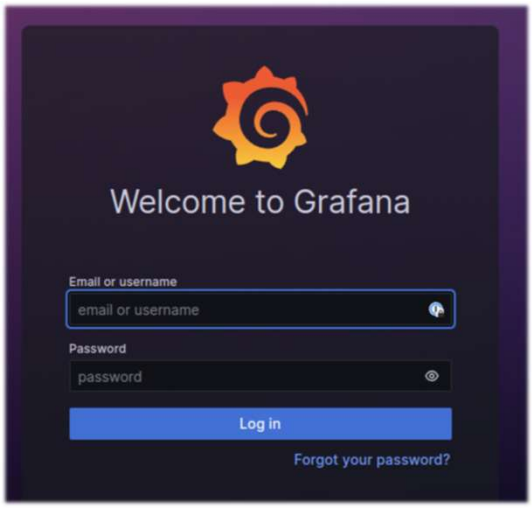
The yaml file has the full specification of the containers in the pod, and **podman play** can rebuild the setup when necessary.

```
[rob@a3530038 ~]$ podman play kube vmprf.yaml
Pod:
b1de4dca7674424a98440b8d184220eaf772c32027aa7eaf305483ec2eb128fd
Containers:
f795aadeb18a35403011826d9f2a059e70fff6193da55b4030a247658d4a6ad1
09f53b06969e01a7616fb07d7f47300d354b996f6afe393692a7fdeafcc142ff
```


You may find it helpful to generate a yaml file with the full set of containers in the pod, including all the parameters and settings that you used. Not just for documentation, but also to use with **podman play** to rebuild the entire pod when necessary. The permissions and other configuration aspects are kept in the persistent volumes that you mount into the containers.

When you're ambitious, go ahead and use **podman pod stop vmprf** and **podman pod rm vmprf** to eliminate what you did, and use **podman play kube vmprf.yml** to create the pod and containers again. Since the configuration is in the persistent volumes, you don't have to repeat that part.

Connect to port 3000



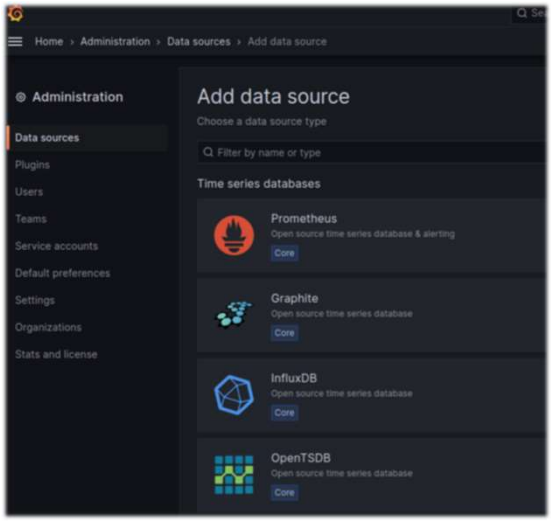
Login with credentials as stated when you created the container

Select the tile to define a data source

Point your browser to port 3000 on the Linux host, which should give you the Grafana login screen.

One the welcome screen, click the 2nd tile to create your first data source.

Create InfluxDB Data Source



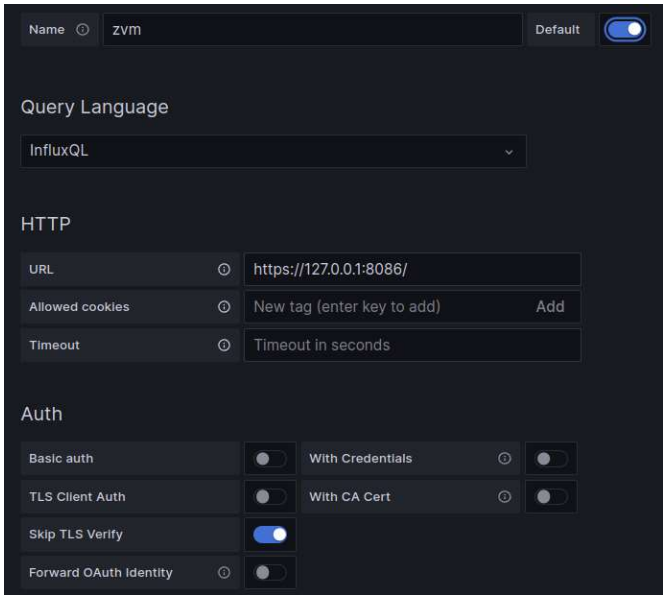
Select an ‘InfluxDB’ data source that is one of the built-in data sources for Grafana

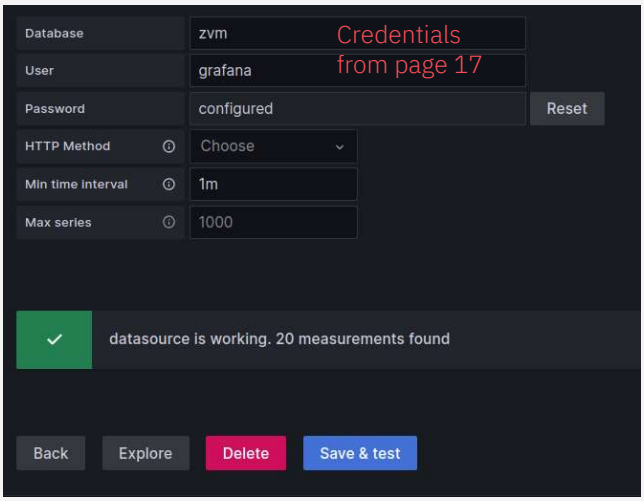
Provide Data Source Details

Specify database name and minimum time interval

Use credentials as defined for Grafana read access

Tick “Skip TLS Verify” when using internal CA





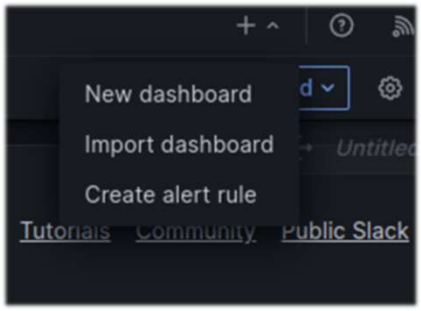
We suggest to name the data source ‘zvm’ in Grafana. This is a local name Grafana, and simply used to reference the data source or connection profile that we are defining.

The URL filled in is http://127.0.0.1:8086/ because we connect to the InfluxDB container that is in the same pod.

When you use an internal certificate authority, tick the button to Skip TLS Verify because the container does not have the root certificate installed that can be used to verify the authenticity of the InfluxDB certificate. It is possible to make this work if you have the loopback address 127.0.0.1 as valid IP address in the certificate.

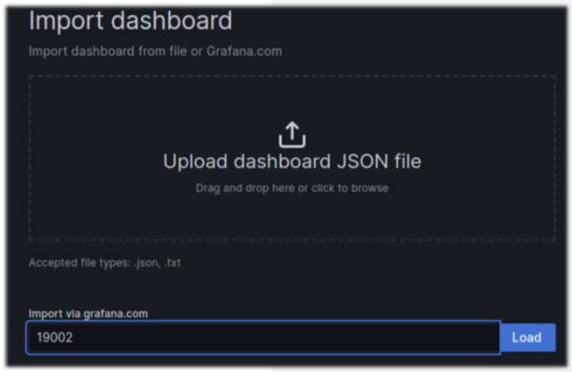
The other thing to fill in is the database name, which is the ‘zvm’ that we created earlier in the InfluxDB container. Since the z/VM monitor by default produces samples every minute, it is wise to specify that here. Once you click the “Save & Test” button, you should get the confirmation that the data source is working, and that measurements are found (the number may vary depending on your configuration). If you see “0 measurements found” that confirms the InfluxDB connection but suggests that the Data Pump is not yet running.

Import a Dashboard



From the Welcome screen, use the [+] button in the upper right to select "**Import dashboard**"

Use "**Import from grafana.com**" when your Linux host can download files from the Internet



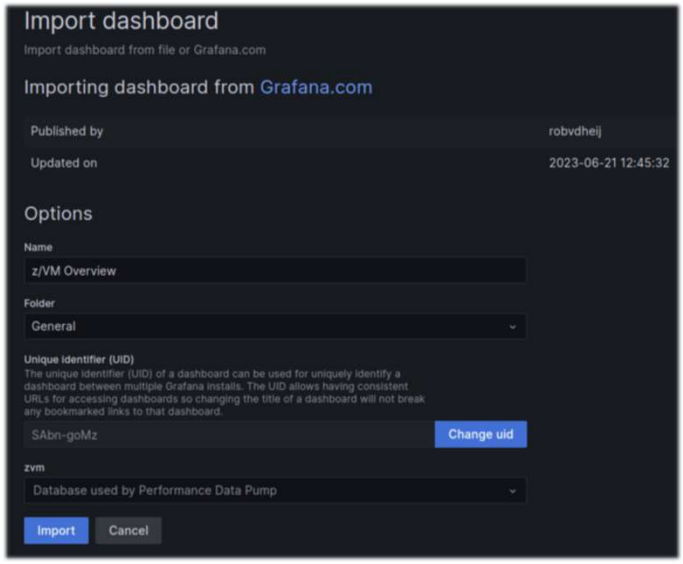
ID: 19002

https://grafana.com/grafana/dashboards/

On the Welcome screen, the [+] in the upper right has a pull-down menu with the "Import dashboard" option. When your Linux server has the ability to download files from the Internet, you can simply select the dashboard with ID 19002 and press [Load].

Without an Internet connection, first download the dashboard to your workstation, and then upload to the "Import dashboard" dialog. Search on https://grafana.com/grafana/dashboards/ for z/VM and select the "z/VM Overview" dashboard. Use [Download JSON] to download a copy of the dashboard.

Import Dashboard (2)

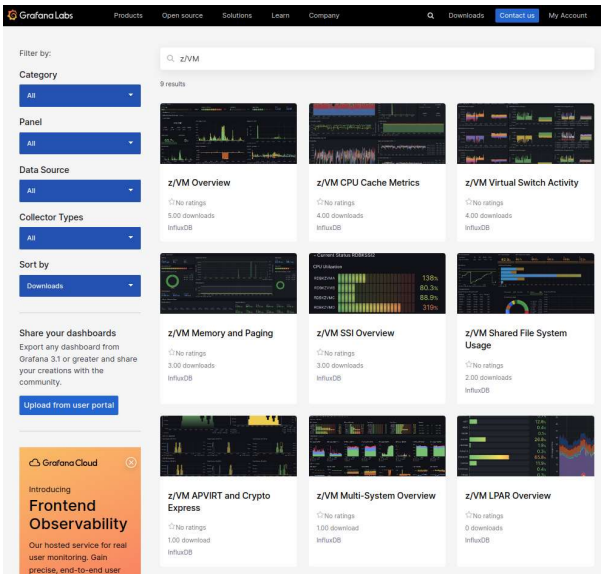


Select the “zvm” data source from the pull-down menu at the bottom of the dialog, and confirm with the [Import] selection

The dashboard should appear next, filled with data

Once you load or upload the dashboard, the second step of the dialog confirms which dashboard you selected. At the bottom of the dialog window, use the pull-down menu to select the data source that you defined earlier. Confirm with the [Import] button and Grafana will display the imported dashboard with filled in data already.

Import More Dashboards

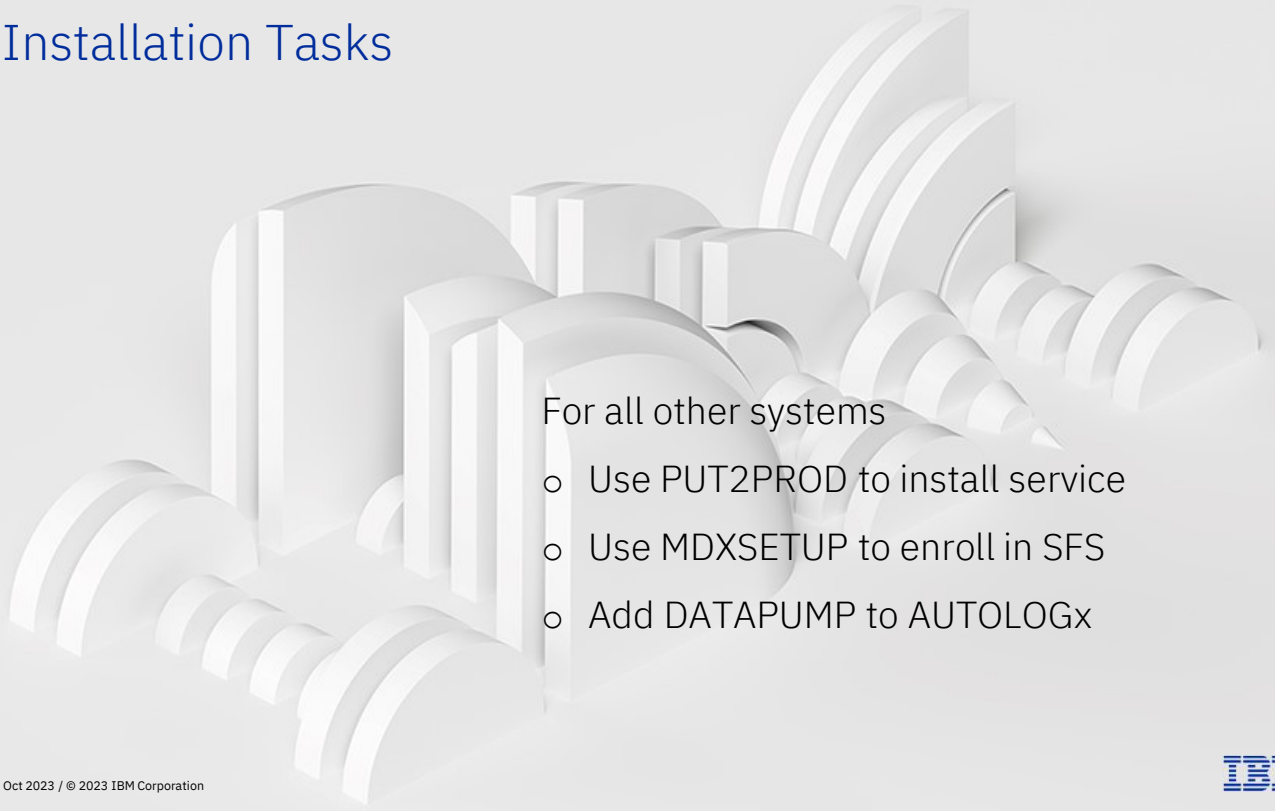


Check the Grafana Labs web site for any additional dashboards that could be useful to import

Search for “z/VM” to see the dashboards that use the Data Pump metrics

https://grafana.com/grafana/dashboards/

Post-Installation Tasks



For all other systems

- Use PUT2PROD to install service
- Use MDXSETUP to enroll in SFS
- Add DATAPUMP to AUTOLOGx

For all other systems

Logon to MAINT730

```
put2prod
 . . .
VMFSET2760I VMFSETUP processing completed successfully (RC=0)
VMFP2P2760I PUT2PROD processing completed successfully (RC=0)
Ready; T=65.39/73.27 17:15:15

vmlink perfsvm 201 ( invoke mdxsetup
DMSVML2060I PERFSVM 201 linked as 0120 file mode Z
Updating files:
DEFAULT  DATAPUMP  2023-08-16 15:44:14 VMPSFS:DATAPUMP.
PROFILE  EXEC      2023-08-16 14:53:53 VMPSFS:DATAPUMP.
Customize configuration files with VMLINK .DIR VMPSFS:DATAPUMP. ( FILEL
DMSVML2061I PERFSVM 201 detached
Ready; T=0.04/0.06 17:17:10
```

When using the same InfluxDB service, there is no additional customization needed

```
xautolog datapump
Command accepted
Ready; T=0.01/0.01 17:18:47
AUTO LOGON  ***       DATAPUMP USERS = 17
HCPCLS6056I XAUTOLOG information for DATAPUMP: The IPL command is verified by the IPL command processor.
```

Data will automatically show in the dashboards

Add DATAPUMP to AUTOLOGx configuration

When you're logging on to MAINT730 on all the other systems, run MDXSETUP as well to enroll DATAPUMP in the local VMSYS: file pool, and copy the configuration files and PROFILE EXEC from the shared VMPSFS file space to the one in VMSYS: for DATAPUMP to use.

When you XAUTOLOG DATAPUMP, it should connect to *MONITOR and start feeding metrics into InfluxDB. Within a few minutes, you will find the pull-down selection the dashboard showing the other systems as well.

Create Service Startup

Generate service files for system

```
[rob@a3530038 ~]$ podman generate systemd -f --name vmprf
/home/rob/pod-vmprf.service
/home/rob/container-influxdb.service
/home/rob/container-grafana.service

[rob@a3530038 ~]$ mkdir -p ~/.config/systemd/user/
[rob@a3530038 ~]$ mv *.service ~/.config/systemd/user/
```

Enable the service

```
[rob@a3530038 ~]$ export DBUS_SESSION_BUS_ADDRESS="unix:path=/run/user/1001/bus"
[rob@a3530038 ~]$ systemctl --user daemon-reload
[rob@a3530038 ~]$ systemctl --user enable pod-vmprf
Created symlink /home/rob/.config/systemd/user/default.target.wants/pod-vmprf.service → . .
[rob@a3530038 ~]$ systemctl --user enable container-grafana
Created symlink /home/rob/.config/systemd/user/default.target.wants/container-grafana.service →  . .
[rob@a3530038 ~]$ systemctl --user enable container-influxdb
Created symlink /home/rob/.config/systemd/user/default.target.wants/container-influxdb.service → . .
```

Reboot to verify containers are started

When you have arranged that your Linux guest on z/VM is restarted after an IPL, you will also want the containers to be started automatically.

With **podman generate** you can create the **systemd** service files that coordinate this. Create the service files and move them to the user directory.

Next, use **systemctl --user** to enable to service. Reboot the guest to verify that the containers are started.