ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOMPL009

CONTROL SYSTEM VIRTUALIZATION AT KARLSRUHE RESEARCH ACCELERATOR

W. Mexner*, B. Aydt, D. Hoffmann, E. Bründermann,
E. Blomley, M. Schuh, A.-S. Müller, Karlsruhe Institute of Technology, Karlsruhe, Germany
S. Marsching, aquenos GmbH, Baden-Baden, Germany

Abstract

With the deployment of a Storage Spaces Direct hyperconverged cluster in 2018, the whole control system server and network infrastructure of the Karlsruhe Research Accelerator have been virtualized to improve the control system availability. The cluster with 6 Dell PowerEdge R740Xd servers with 1,152 GB RAM, 72 cores and 40 TB hyperconverged storage operates 120 virtual machines in total. We will report on our experiences running EPICS IOCs and the industrial control system WinCC OA in this virtual environment.

INTRODUCTION

The Karlsruhe Research Accelerator (KARA) at KIT [1] is a 110m 2.5 GeV electron storage ring with a 53 MeV microtron and 500 MeV booster. The design of the KARA control system is based on EPICS 7.0 with Control System Studio (CSS) as the main operators interface to the control system. Due to historical reasons, the beam lines are controlled by TANGO and the commercial SCADA System WinCC OA 3.15 [2].

Already in 2014, first steps in control system virtualization were successfully taken based on KVM at FLUTE [3] and also at KARA with a small Hyper-V Cluster with a network attached storage for the virtual machines (VM). As most of the control system servers had to be replaced in 2017, the decision was taken to move towards a complete virtualization of all servers. The reasons have been:

- Higher control system availability due to the ability to have zero-downtime hardware and base-OS updates through live-migration of VMs
- The ability for a VM to restart on another host in the event of sudden hardware failure
- Automatic deployment of new servers without any hardware depencies and without buying extra servers.
- Easy resource management of storage and RAM.
- Simple hyper-visor based backup routines

On the other hand, virtualization introduces also new pitfalls:

- As it is easy to create new VMs, you can also easily keep preliminary versions of control servers. These old dormant VMs can deviate far from the actual control system baseline causing a lot of trouble, if inadvertently started.
- Automatic migration: It is tempting to configure automatic migration of control servers in case of hardware failures of a virtualization cluster, but if you have not really a high availability cluster, then this means restart of the servers on a new node causing short interruptions of several servers.
- Control over virtual networks: It is very comfortable having also the network virtualized. But changing from one network to the other means only changing a number for the network adapter, shutting down the network, if inadvertently changed.
- Network stability: Your cluster backbone has to be really stable. From experience, some control hardware has a corrupt network stack implemented, working badly with a virtualized environment
- Resource over provisioning: Virtual machines are splitting the maximum I/O of the virtualiziation hosts. Therefore you need to take care for the I/O load of all virtual machines.

HYPER-V CLUSTER CONCEPT

For KARA we started with the traditional concept of a network attached storage (RAID 5 with 16 hard disks, connected with 10 GBit Ethernet) for the VM images and a pool of Hyper-V compute hosts. Quickly we found that in this simple setup with around 60 virtual machines the limiting factor was the I/O capacity of the NAS storage slowing down all virtual machines.

In 2016, Microsoft introduced the concept of Storage Spaces Direct with Windows Server 2016, so we decided to test for the next productive cluster this promising concept. Storage Spaces Direct [4] uses industry-standard servers with local-attached drives to create highly available, highly scalable software-defined storage at a fraction of the cost of traditional SAN or NAS arrays. Its converged or hyperconverged architecture radically simplifies procurement and deployment, while features such as caching, storage tiers, and erasure coding, come together with the latest hardware innovations such as RDMA networking and NVMe drives.

Hardware Concept

The cluster consists of 6 Dell PowerEdge R740Xd servers each having 3.2 TB NVME for cache, 3.2 TB SSD for Hot-Data Tier and 16 TB HDD for Cold Data and a Intel Xeon Gold CPU with 12 cores, resulting in a final cluster capacity of 1,152 GB RAM, 72 cores and 40 TB storage (see Fig. 1).

In 2019, the control group faced the problem, that the used version of WinCC OA (3.11) was discontinued and not anymore running under modern operating systems. In addition, the newest version 3.15 was incompatible to the

[*] wolfgang.mexner@kit.edu

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOMPL009

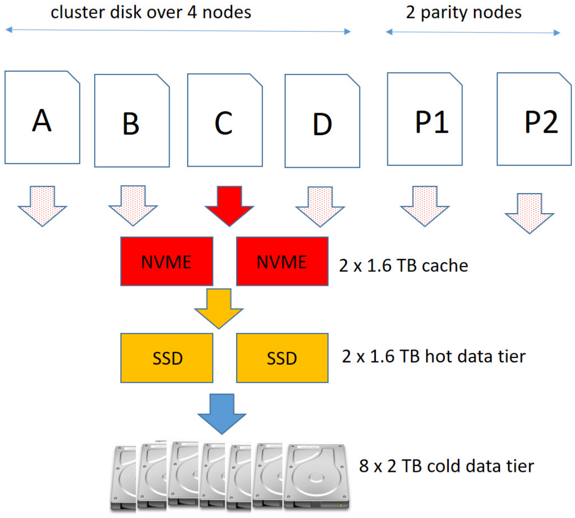

Figure 1: Cluster hardware concept

old version, therefore a control system upgrade for all 18 beamlines in a single shutdown was required. In this situation, it was decided to move to a fully virtualized control system environment for all beamlines. Instead of installing new control system servers at each beamline, all originally separate private networks for each beamline have been virtualized and connected with 10GbE to the new Hyper-V Cluster (see Fig. 2). The control server has been configured once and cloned once for each beamline with a dedicated configuration file and virtual network interface. The whole SCADA communication was sped up by a cluster internal network between all servers.

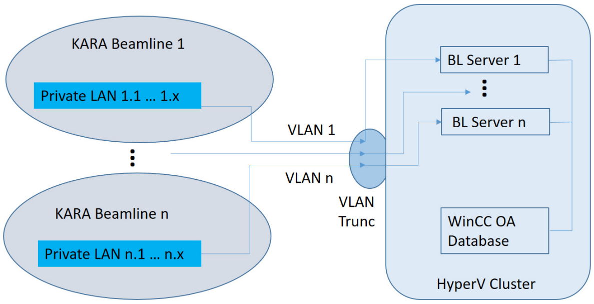

Figure 2: Beamline virtualization concept

Licensing

As it is a commercial software, WinCC OA requires USB dongles for operation. This problem was solved by an external USB dongle server, allowing to attach the USB dongle virtually to a selected VM.

Networking

The internal cluster network is handled by a QLogic network board equipped with two redundant RDMA capable 10GbE ports, which are configured as a team. So the whole internal cluster network is handled by 12 separate 10GbE ports. For external communication, two 10GbE ports are configured as a team as well connecting all virtual networks (VLANs) to the internal Hyper-V fully virtualized network switch, allowing to connect each VM to every network required, also when a VM is transferred from one node to another.

CONTROL SYSTEMS VIRTUALIZATION WINCC OA

Structure

WinCC OA Version 3.15 is the overall Supervisory Control and Data Acquisition System (SCADA) for all KARA beamlines (see Fig. 3). WinCC OA is the interconnection between the TANGO based beamline and the EPICS based KIT synchrotron control. The beamline control with SPEC as command UI and WinCC OA as graphical UI are connected via an own developed TANGO interface to the SCADA system.

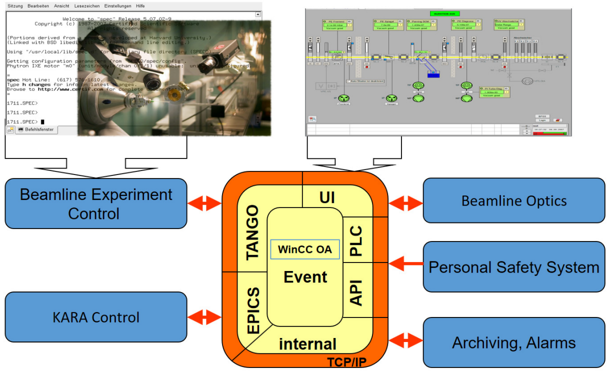

Figure 3: WinCC OA SCADA Concept

Performance Constraints

The central WinCC OA SCADA server is logging around 4000 values with 1 Hz to disk, resulting in a continous I/O load to the virtualization server. Periodically additonal load is caused by the VEEAM backup. As this was causing some latency, the VM of the central server was moved to a replication node. This replication node is a separate virtualization server, which is permanently cloning all hard disk changes to the central cluster disk. So in case of a failure of this external server, Hyper-V is performing an automatic failover restarting the server inside the cluster. By this the central server has the full I/O capacity of a barebone server, but still the advantages of hardware virtualization.

Introducing Docker

A typical control VM requires around 40 GB of harddisk space plus around 4 GB of cluster RAM. In a first test we found that WinCC OA is smoothly integrating into a Docker environment with portainer as graphical Docker management tool. Such a docker image requires only 1 GB of RAM and can be handled much more much easily than a full virtual machine. At the moment we are preparing the Docker images for the first beamlines.

EPICS

Server Concept

Most EPICS servers (IOCs) run inside the virtualized environment. This is possible as most hardware is nowadays controlled via Ethernet (or serial interfaces that can be converted to Ethernet). The notable exception to this rule are the beam-position monitors, the low-level RF system, the bunch-by-bunch feedback system, and the data acquisition devices, which all run EPICS IOCs directly on embedded computers that are built into these devices.

The virtual environment for the EPICS servers is provided by three virtual machines: one for IOCs running in production mode, one for Java-based servers related to EPICS services (e.g. the Control System Studio alarm server), and one for testing new IOCs. This setup has proven sufficient as in a virtualized environment, which can provide sufficient CPU and memory resources, there is no gain in distributing IOCs across several computers.

Issues with Link Aggregation

As mentioned earlier, we use link-layer aggregation on the virtual machine hosts in order to improve both performance and availability. While this works fine for the network interfaces that are part of the I/O backbone of the cluster, it causes unanticipated problems when being used for the network interfaces to the “outside world”.

These problems are caused by a significant number of embedded devices, which have a wrong implementation of the TCP/IP stack. Due to the way link aggregation of the bonded VM host network adapters (NICs) works, a different source MAC address is used depending on which of the bonded NICs the packet is sent. When using the first adapter, the packets originate from the MAC address of the VM as expected. However, when using the second adapter, the packets originate from the MAC address of the network interface in the host system. This means that packets sent by the same VM will randomly have two different source MAC addresses. By default these devices should use the address resolution protocol (ARP) to resolve the source IP address of a packet back to a destination MAC address. Instead, they simply send the response back to the originating MAC address. Hyper-V, however, discards packets that are targeted at the IP address of a VM, but specify the destination MAC address of the host system.

In theory, one should be able to resolve this issue by setting the load-balancing mode to “Hyper-V port” or using the LACP instead of the static teaming mode [5]. The first option did not seem to reliably fix the problems for us and the second option did not work reliably either, seemingly because of some compatibility problems with the Cisco network switches that we use.

For this reason, we had to implement a workaround. Instead of directly communicating with devices that are affected by these problems, we use a Raspberry Pi minicomputer that acts as a reverse proxy between these devices and the Hyper-V virtual machine. This fixes the problems completely.

AUTOMATIC SERVER DEPLOYMENT

One of the goals of virtualization is making the operation of IT systems more efficient by reducing the number of hardware systems that need to be deployed and maintained. However, considerable time and effort is still needed for the deployment and maintenance of systems installed inside the virtual machines. In order to further reduce this effort, tools that automate the deployment of operating systems and software and help with keeping the systems’ configurations up to date are needed. Very much like virtualization helps with reducing cost by reducing the number of “real” hardware systems, automation tools help by effectively reducing the number of configuration profiles that need to be managed manually.

Initial Deployment

Originally, we used Cobbler [6] for automating the installation of Ubuntu Linux systems (both inside virtual machines and on real hardware systems). However, we quickly became unhappy with how Cobbler worked. In particular, it lacked sufficient support for customizations and was hard to integrate into our existing environment of network management tools. Another issue was that it had a signficant number of bugs and was poorly maintained. For example, no Debian packages were provided for recent versions and such packages could not be easily built from the source tree either. For these reasons, we developed our own solution called Vinegar [7].

Vinegar is entirely written in Python and integrates a HTTP and TFTP server. Using the Jinja template engine, it first renders configuration files for GRUB 2, delivering them over the TFTP server. Subsequently, the Linux kernel and initial ramdisk of the installer system are loaded from the TFTP server by GRUB, and the installer system then loads its configuration files (again templates rendered with Jinja) via HTTP. This setup has proven to be very easy to use and integrates perfectly with our existing environment.

Configuration Management

Once the operating system is installed, one still has to manage the system configuration (installed software components, configuration files, etc.). In contrast to deploying a new system, this typically is a recurrent task as the target configuration of a system is a moving target due to software updates and configuration changes.

We evaluated four different solutions for managing configurations: Ansible, Chef, Puppet, and SaltStack. Chef and Puppet are written in Ruby while Ansible and SaltStack are written in Python. While we already had significant experience in developing and maintaining software written in Python, we had no such experience in Ruby. This is why we discarded Chef and Puppet early on. While Ansible and SaltStack have some similarities, Ansible seems to be a bit more limited: It exclusively uses SSH to manage systems, which makes it use for systems running Windows more difficult and limits the performance. SaltStack, on the other hand can use SSH like Ansible, but it also has an integrated high-performance transport based on ZeroMQ, that provides superior performance when managing a large number of systems and works on Windows without any hassle.

For these reasons, we decided to go with SaltStack [8] and have been using it for about three years without ever looking back.

SaltStack is automatically installed on target systems as part of the automated installation of the operating system. Once installed, a system automatically appears in SaltStack and subsequently, configuration profiles can be assigned. We use the configuration profiles for a number of purposes, e.g. managing the DNS and DHCP servers of the control-system network, managing the operator consoles (installing Control System Studio and keeping panels up to date), and managing EPICS servers (installing the EPICS environment, checking out EPICS IOCs from the source code repository and compiling them). Besides reducing the time and effort needed to maintain these systems significantly, SaltStack has brought us other advantages: The configuration profiles in Salt also serve as a centralized documentation of the system configuration (due to the possibility to have comment sections inside the profile definitions) and by having centralized configuration profiles, one never has to worry again about whether a configuration change has really been applied to all affected systems.

At the moment, we are looking into also using SaltStack to manage the configuration of some embedded devices: SaltStack comes with a component called Salt Proxy that allows managing devices that cannot run Salt itself. This component comes bundled with a number of plugins (e.g. for managing network switches via SNMP) and we are looking into writing our own plugin so that it can be used to manage our data-acquisition systems from D-TACQ.

CONCLUSION

We successfully virtualized the EPICS and WinCC OA control systems servers for KARA and the beamlines operated at the KIT synchrotron using a Hyper-V storage spaces direct cluster. Vinegar and StaltStack are powerful tools for the automatic server deployment.

REFERENCES

[1] Karlsruher Research Accelerator KARA, https://www.ibpt.kit.edu/kara

[2] WinCC OA, https://www.winccoa.com

[3] W. Mexner *et al.*, "Update on the Status of the FLUTE Control System", in *Proc. PCaPAC'18*, Hsinchu City, Taiwan, Oct. 2018, pp. 54-56. doi:10.18429/JACoW-PCaPAC2018-WEP10

[4] Storage Spaces Direct, https://docs.microsoft.com/en-us/windows-server/storage/storage-spaces/storage-spaces-direct-overview

[5] E. Siron, "Fixing Erratic Behavior on Hyper-V with Network Load Balancers", https://www.altaro.com/hyper-v/erratic-behavior-hyper-v-network-load-balancers/

[6] Cobbler, https://cobbler.github.io/

[7] Vinegar, https://github.com/KIT-IBPT/vinegar/

[8] SaltStack, https://www.saltstack.com/