Performing Diagnostics on an NVIDIA GB200 GPU Pre and Post-Repair

Problem statement

In high-performance computing environments, such as data centers, the NVIDIA GB200 GPU plays a critical role in handling complex computations and workloads. However, like any advanced hardware, GPUs can suffer from a range of issues such as overheating, memory errors, power inefficiencies, NVLinks or PCIe communication failures. These problems can lead to degraded performance, system instability, and costly downtime, particularly when dealing with mission-critical applications.

Therefore, there is a need for a comprehensive, automated diagnostic probe that can perform in-depth pre-repair diagnostics to identify hardware and software issues and conduct post-repair validation to ensure the system is fully functional and operating before handing them off to the customer.

Key Challenges:

- **Pre-Repair Diagnostics**: Without a systematic probe, it's difficult to accurately diagnose various issues related to hardware, driver, thermal performance, memory faults, power constraints, NVLink or PCIe bandwidth degradation. This makes it challenging to pinpoint the exact root cause of failures.
- **Post-Repair Validation**: After performing repairs, technicians need a reliable way to confirm that all issues have been resolved. Manual checks may not detect subtle or intermittent problems that could recur, leading to system crashes or poor performance before handing the device back to the customer.

When diagnosing and repairing an NVIDIA GB200 GPU, a systematic approach using the **Partnerdiag** tool desgined for NVIDIA's hardware partners or a combination of tools available to general users such as ***nvidia-smi*****, nvvs,** ***dcmg*** and **Redfish API**, can ensure thorough diagnostics before and after repairs. These tools offer granular insights into the GPU's health, performance, and potential issues that might affect its operation in real-world applications. This document provides an overview on how these commands can be used in the context of pre-repair troubleshooting and post-repair validation.

After the initial hand-off of the systems to OAI, we would need a password to access the ILOM via SSH. OCI will not have the password after the hand-off therefor OAI team needs to run the probes on our behalf.

The focus of this document is what information the probe must collect to aid in the diagnostics efforts.

Proposed Solution

***As of 9/23/24 we don't have access to a GB200 systems, and much of this information is collected from reading various NVIDIA documentations.*

We can provide a dedicated probe(s) that uses diagnostics tools to run diagnostics at computer tray, switch tray and rack level:

- **Pre-Repair**: Automatically identify and log critical issues such as overheating, memory errors, power inconsistencies, NVLink, PCIe and GPU throttling and failures.

- **Post-Repair**: Verify that the repairs have successfully restored the GPU to optimal performance, confirming that temperature, power, memory, NVLink, and PCIe communication are functioning correctly before handing the device back to the customer.

By implementing this probe, we can:

1. **Increase Repair Efficiency**: Provide detailed diagnostics that guide technicians in addressing the correct issues during repair.
2. **Ensure Performance Stability**: Validate the repair's success and prevent future failures by ensuring the GPU operates within safe and stable thresholds post-repair.
3. **Potentially Minimize Downtime**: Quickly identify problems before they lead to complete system failures. This is only possible if we are allowed to run the probe in the background similar to a canary.

Suggested Pre-Repair Diagnostics to identify issues

1. Run *Partnerdiag*, nvvs or *`dcgmi healthcheck`* commands to check critical health metrics.
    a. Run a light GPU load test to detect crashes or throttling.
2. Monitor GPU and memory temperatures using *`nvidia-smi`*.
3. Verify power consumption and clock speeds for throttling or power issues.
    a. Verify stable power delivery by checking power draw and limits.
4. Check for
   ECC memory errors to identify VRAM issues.
5. Scan kernel logs for *Xid driver errors* to uncover specific GPU failure points.
6. Test NVLink and PCIe bandwidth to ensure proper communication between GPU and system. These test also should include PRBS Pseudo-random bit sequence).
7. Check cooling system (fans or liquid cooling) to ensure adequate thermal performance.
    a. Use *Partnerdiag,* nvidia-smi or *`ipmitool`* for system temperature monitoring to confirm cooling and power stability at the system level.

Suggested Post-Repair Diagnostics to Verifying the Fix

1. Perform another **Partnerdiag***,* **nvvs** or **`dcmg`** `healthcheck` to verify that all critical health metrics (temperature, memory, power) are within acceptable ranges post-repair.
2. Check the GPU's temperature again under load. A stable temperature below 85°C, especially after replacement of any faulty hardware components. (e.g. fixing cooling issues) and confirm the repair is successful.
3. Use **Partnerdiag, nvvs** or **`dcmg`** to perform a stress test on the GPU, loading it fully to ensure it can handle maximum workloads without throttling or overheating.
4. Monitor power consumption and ensure that the GPU's clock speeds are stable and operating near their rated values under load.
5. Check that no new **ECC errors** are detected post-repair, confirming that memory issues have been resolved.
6. Check the N**VLink or PCIe bandwidth** to confirm the GPU is communicating correctly with the rest of the system. Also to stress the NVLink interconnects for stability and errors.
7. Scan the kernel logs one more time to ensure no new **Xid driver errors** are being logged, indicating the GPU is stable.

Open quesitons:

Can we leverage HPC Doctor for pre and post repair process? (code)

As of 10/1/24 the HPC Doctor does not support GB200. However, it might be a good idea to coordinate our efforts as it seems that our activities are similar. Some impediments to using the HPC Doctor might be the lack of compute as it is needed to talk to pulse server or HOPs.

Proposed Diagnostics Probes

Tools:

nvidia-smi: (NVIDIA System Management Interface) is a command-line tool provided by NVIDIA for monitoring and managing NVIDIA GPU devices. It is included in the NVIDIA GPU drivers and is primarily used in Linux and Windows environments. `nvidia-smi` provides detailed information about the status and performance of NVIDIA GPUs, making it essential for developers, system administrators, and data scientists working with GPUs in workstations, servers, or data centers. RUST lib to do diagnostics https://docs.rs/nvml-wrapper/latest/nvml_wrapper/ Rust wrapper for the NVIDIA Management Library (NVML), a C-based programmatic interface for monitoring and managing various states within NVIDIA GPUs. It provides a direct access to the queries and commands exposed via nvidia-smi. Similarly we have the **Go NVML bindings**: `go-nvml`.

nvvs: NVIDIA Validation Suite (also referred to as DCGM Diagnostic). It is a tool provided by NVIDIA as part of the Data Center GPU Manager (DCGM) suite. The primary purpose of NVVS is to validate and diagnose the health and performance of NVIDIA GPUs, particularly in data center environments. There isn't a direct **Rust** or **Go** library that provides the exact same functionality as NVIDIA's **NVVS** (NVIDIA Validation Suite).

nvqual: NVQual is a command-line tool that is part of the **NVIDIA Data Center GPU Manager (DCGM)** suite. It is designed to perform hardware qualification tests on NVIDIA GPUs in data center environments.

nvrastool: this tool can be used as a wrapper to run a set of tests on the GB200 NVLalso can be sued to inject faults for testing purpuses. it can act as a wrapper for the Redfish API calls to run a set of tests on the NVSwitch and most likely on the GPU Server as well. (ref: NVRASTool User Guide v1.1.pdf)

dcgmi: A command-line interface (CLI) for **NVIDIA Data Center GPU Manager (DCGM)**. DCGM is a suite of tools designed to monitor, manage, and diagnose NVIDIA GPUs, particularly in large-scale data centers where GPU health, utilization, and performance are critical. I was not able to find any **Rust** or **Go** libraries that provide full equivalent functionality to **`dcgmi.`**

Redfish API: A standard RESTful interface designed for managing and interacting with servers, storage, networking, and other infrastructure devices. It provides a modern, secure, and scalable way to manage hardware and systems in data centers.

*****Redfish API** can be **partially used** to monitor system health, power consumption, and fan speeds, but most **GPU-specific diagnostics** (like temperature, ECC errors, clock speeds, NVLink tests, and stress testing) must be done using **nvidia-smi**, **dcgmi**, or **nvvs**.*

Partnerdiag: It is a more advanced tool designed for **NVIDIA's hardware partners**. It provides **deeper system-level diagnostics** and integration that might not be necessary for regular users but are essential for vendors and large-scale data centers. It provides Temperature monitoring, ECC Memory Errors, Power Consumption clock speed stability and NVLink Heath diagnostics. This tool is most likely is provided as a tarball file and it must be extracted and executed on the system under test. The switch tray diagnostics must be executed in the NVOS environment on the switch tray and the compute tray diagnostics must be executed in the host OS. The **Module and Orchestration Diagnostic Software**, and the **Secure Partition** must be enabled on the system under test. (It is not clear if we can have this enabled always as enabling it requires a system reboot and during per-repair we can lose important data during any reboot). Secure Partition is a secure and isolated environment within the system designed specifically for running diagnostics and performing stress tests on NVIDIA hardware components, such as GPUs.

nvdebug: The NVIDIA debug tool runs on server platforms and from remote client machines. we can use it to collect summary reports, log analysis and data. we can use nvdebug to collect all the logs from the Redfish collector (e.g. *nvdebug -i $BMC_IP -u $BMC_USER -p $BMC_PASS -t $TARGET -g Redfish*). or colelct logs from the NVSwitch (e.g. ./nvdebug -t NVSwitch -l Host). additional information can be found in the "NVIDIA Debug Tool - user Guide.

Field diag: GB200 SEA5 Rack 0116 Bring-Up Notes

NVSSVT: simultaneously perform health and other checks on the GPU Servers and vswitch tray nodes on multiple GB200 systems at once.

```
nvssvt -c config.yaml -d dut_config.yaml -q R1 -s MGX-arm64 -b "GB200 NVL". -m "SystemManagement" "Telemetry"
"HostBasedReadiness"
nvssvt -c nvswitch_config.yaml -d dut_config.yaml -q R1 -s DGX-arm64 -b "GB200 NVL NvswitchTray" -m
"SystemManagement" "Telemetry" "HostBasedReadiness"
```

GB200 Manual Run Outputs

1	Command	*partnerdiag* ***as of 10/1/24, no public documentation*** ***found, we may have access to this doc internally.***	nvidia-smi (**should use --format=csv option for more compact results.) Also, similar API available in RUST & GO.	NVVS (NVIDIA Validation Suite) ***can use csv or json output:*** ***`--output-format=csv`*** ***`--output-format=json`***	DCGMI (Data Center GPU Manager Interface)	NVQual	other	Redfish API ***Redfish is useful for broader system monitoring, but it may lack the fine-grained detail and focus that `nvidia-smi` provides for NVIDIA GPUs.*** ***** As of 9/23/24, this data is mostly about older version because I could not find anything on GB200.***	Pre or Post Repair	Recommend for GB200

1	**GPU Health Check**	yes sudo ./partnerdiag --level1 level 2 --gpufielddiag, --log <path> to set the bios before running partnerdiag: BIOS Automation sudo ./partnerdiag --field --level1 --run_on_error --no_bmc	nvidia-smi --query-gpu	nvvs -t healthcheck	dcgmi health -g 0	nvqual --gpu <gpu_index> --test <test_type> --duration <duration> looking at the document "Introduction to NVIDIA Partner Validation Playbook V05.pdf" we have the following test IDs:		Basic hardware health	Pre and Post	dcgmi for data c nvidia-smi for standalone

#	ID	Category
1	**CL**	**Clock**
2	**DG**	**Diagnostics**
3	**EC**	**PCIe**
4	EM	EMC
6	GP	GPIO, SGPIO
7	IC	I2C, I3C
8	JT	JTAG
9	**LC**	**Liquid Cooling**
10	LT	LTPI (LVDS Tunneling Protocol & interface)
11	MC	Management Controller (HMC, BMC)
13	NC	Network Controller Sideband Interface(RMII_NCSI)
14	**NT**	**Network**
15	**NV**	**NVLink**
..	...	...
19	**PW**	**Power**
22	**SD**	**Storage**
26	**SY**	**System**
27	TEL	Telemetry
28	**TH**	**Thermal**

2	**ECC Errors**	yes	nvidia-smi --query-gpu=ecc.errors.*	nvvs -t ecc	dcgmi diag -g 1 -r 3	Yes		No equivalent Redfish does not provide the following: *Detailed GPU ECC error metrics, while `nvidia-smi` offers a breakdown by memory type and error type.* *Does not report volatile (session-specific) vs. aggregate (lifetime) errors, which are crucial for GPU error monitoring.* *Does not offer error counts broken down by memory type (e.g., L1 cache, L2 cache, device memory).* *Does not offer real-time, detailed error monitoring specifically for GPU memory* GET /redfish/v1/Systems/{system_id}/**PCIeDevices**/{pcie_device_id}	Pre	nvidia-smi or dc (detailed ECC a

3	**Temperature Monitoring**	yes	nvidia-smi --query-gpu=temperature.gpu	nvvs -t short	dcgmi telemetry --interval	Yes	ipmitool sdr list | grep -i temp ** Please review the 'Appendix D' for additional information.	GET /redfish/v1/Chassis/{chassis_id}/Thermal/ `{ "@odata.id": "/redfish/v1/Chassis/1/Thermal/", "Temperatures": [ { "MemberId": "HMC-TEMP-CURR", "Name": "Host Management Controller Temperature", ... ] }` GET /redfish/v1/Systems/{system_id}/Thermal • *`nvidia-smi` provides direct and GPU-specific temperature information, which is precisely the current GPU core temperature.* • *Redfish may report general temperature sensors that include the GPU, but this information might be less granular or tied to broader system monitoring rather than direct GPU telemetry.*	Pre and Post	nvidia-smi for re dcgmi for telem
4	**Power Draw**	yes	nvidia-smi --query-gpu=power.draw, nvidia-smi --query-gpu=power.draw, power.limit	nvvs -t power	dcgmi setpower --gpu	Yes		GET /Chassis/<chassis-id>/Power	Pre and Post	nvidia-smi (simp dcgmi (control c

5	**Clocks Monitoring**	yes	nvidia-smi --query-gpu=clocks.* \ nvidia-smi --query-gpu=clocks.sm,clocks.memory
			`nvidia-smi --query-gpu=clocks.* --format=csv`
			`clocks.gr [MHz], clocks.sm [MHz], clocks.mem [MHz], clocks.video [MHz]`
			`1530, 1530, 877, 540`
				nvvs -t clocks	dcgmi stats --pid	Yes		No equivalent
								***The main difference when calling `nvidia-smi --query-gpu=clocks.*` compared to a Redfish API equivalent is the level of **detail and granularity**. `nvidia-smi` provides highly specific real-time clock speed information for various GPU components (core, memory, SM, video), while Redfish APIs typically offer more general performance or power metrics, and may not consistently expose clock data unless specifically implemented by the hardware vendor.*
								GET /redfish/v1/Systems/{system_id}/GPU/{gpu_id}
									Pre and Post	nvidia-smi for re
										dcgmi for deepe
										profiling

6	**PCIe Bandwidth**	yes	nvidia-smi pcie
			
			`$ nvidia-smi pcie`
			
			`Tx Throughput : 180.00 MB/s`
			`Rx Throughput : 500.00 MB/s`
			`Current PCIe Link Generation : Gen3`
			`Current PCIe Link Width : x16`
				No equivalent	dcgmi diag -g 1	Yes		No equivalent
								
								Redfish doesn't typically provide real-time PCIe bandwidth utilization (Tx /Rx throughput in MB/s), which `nvidia-smi pcie` provides.
								
								*GET /redfish/v1/Chassis/{chassis_id}/**PCIeDevices***
									Pre and Post	nvidia-smi (basi bandwidth chec

7	**Fan Speed**	yes	nvidia-smi --query-gpu=fan.speed	No equivalent	No equivalent	Yes		GET /Chassis/<chassis-id>/Thermal
								
								GET /redfish/v1/Systems/{system_id}/GPU/{gpu_id}
								
								Mock output
								
								`{`
								`    "@odata.context": "/redfish/v1/$metadata#GPU.GPU",`
								`    "@odata.id": "/redfish/v1/Systems/1/GPU/0",`
								`    "@odata.type": "#GPU.v1_0_0.GPU",`
								`    "Id": "0",`
								`    "Name": "NVIDIA Tesla V100",`
								`    "Status": {`
								`        "State": "Enabled",`
									Pre and Post	nvidia-smi (for s fan monitoring)

```
"Health":
"Warning"
    },
    "Fan": {

"SpeedRPM":
4500,

"Status": {

"State":
"Enabled",

"Health":
"Warning",

"Message":
"Fan speed
is above
normal
range."
        },

"Errors": {

"FanFailure"
: false,

"Message":
"Fan
operating
within
acceptable
limits, but
requires
attention."
        }
    },

"Temperature
": {

"Current":
80,

"Max": 90
    },
    "ECC": {

"Errors": {

"Uncorrected
Total": 3,

"CorrectedTo
tal": 15
        }
    }
}
```

8	**Memory Usage**	yes	nvidia-smi --query-gpu=memory.* memory.total,	nvvs -t memorycheck	dcgmi diag -g 1	Yes		GET /redfish/v1/Systems/{system_id}/Memory	Pre and Post	nvidia-smi or NV memory diagnos

			memory.used memory.free ecc.errors. volatile.corrected **ecc.errors. volatile. uncorrected ecc.errors. uncorrected_total,** ecc.errors. corrected_total					curl -X GET \ -H "Content-Type: application/json" \ **https://<our_redfish_endpoint>/redfish/v1/Systems/{system_id}/GPU/{gpu_id}** \ -u "username:password" **Mock output** `{ "@odata.context": "/redfish/v1/$metadata#GPU.GPU", "@odata.id": "/redfish/v1/Systems/1/GPU/0", "@odata.type": "#GPU.v1_0_0.GPU", "Id": "0", "Name": "NVIDIA Tesla V100", "Status": { "State": "Enabled", "Health": "OK" }, "ECC": { "Errors": { "UncorrectedTotal": 5, "CorrectedTotal": 20 } }, "Memory": { "TotalMB": 16384, "UsedMB": 8192 }, "Temperature": { "Current": 75, "Max": 90 } }`	

9	**Stress Test**	yes	No equivalent	nvvs -t stress --output-format=csv
				
				`gpu_id, status, message`
				`0,PASS, Stress test completed successfully.`
				`1,PASS, Stress test completed successfully.`
				
				Remote execution via ssh. Please look at the appendix E) 'Stress tesing remotely ' for additional information on running nvvs remotely using ssh.
					dcgmi stress --gpu
					
					Ensuring that the GPUs can handle workloads under extreme conditions.
					
					Remote execution via ssh.
						Yes		No equivalent
								
								*The **Redfish API** is designed primarily for **system-level management** and lacks the ability to perform GPU-specific stress tests like `dcgmi stress --gpu` or `nvvs -t stress`. It focuses more on monitoring and managing hardware (power, thermals, etc.) but does not provide mechanisms to stress-test GPUs directly.*
								
								curl -X GET \ -H "Content-Type: application/json" \ **https://<our_redfish_endpoint>/redfish/v1/Systems/{system_id}/TestResults** \ -u "username:password"
								
								mock output
								`{`
								`    "@odata.context": "/redfish/v1/$metadata#TestResults.TestResults",`
								`    "@odata.id": "/redfish/v1/Systems/1/TestResults",`
								`"TestType": "Stress",`
								`"Results": [`
								`        {`
								`"gpu_id": "0",`
								`"status": "FAIL",`
								`"message": "Temperature exceeded safe limits.",`
								`"error_code": "TEMP_TOO_HIGH",`
								`"duration_seconds": 300,`
								`"start_time": "2024-09-24T10:00:00Z",`
								`"end_time":`
									Post	NVVS or dcgmi post-repair stres validation)

								`"2024-09-24T10:05:00Z"`
								`        },`
								`        {`
								`"gpu_id": "1",`
								`"status": "PASS",`
								`"message": "Stress test completed successfully.",`
								`"duration_seconds": 300,`
								`"start_time": "2024-09-24T10:00:00Z",`
								`"end_time": "2024-09-24T10:05:00Z"`
								`        }`
								`    ],`
								`"timestamp": "2024-09-24T10:05:00Z",`
								`"overall_status": "FAIL",`
								`"summary": "One or more tests failed."`
								`}`		

10	**NVLink Errors**	yes (bandwidth test)	nvidia-smi nvlink --format=csv
			`# sample output:"Source","Destination","Bandwidth (GB/s)","Link State","Link Type"`
			`"GPU 0","GPU 1","25.0","UP","NVLink"`
			`"GPU 1","GPU 2","25.0","UP","NVLink"`
			`"GPU 0","GPU 2","25.0","DOWN","NVLink"`
				No equivalent	dcgmi nvlink --status.	Yes	dmesg | grep -i nvlink
							
							dmesg | grep -i xid
							
							** xid errors could have NVLink errors.
								No equivalent
								
								While there isn't a single Redfish API call that replicates the exact functionality of `nvidia-smi nvlink`, we can gather similar interconnect and link information by using a combination of Redfish API calls, but we need to test this out on the BG200 hardware to verify it support similar output.
								
								GET /redfish/v1/Systems/{system_id}/GPU/{gpu_id}
								
								*GET /redfish/v1/Systems/{system_id}/**Links***
								
								*GET /redfish/v1/**Chassis**/{chassis_id}*
								
								*GET /redfish/v1/**Managers**/{manager_id}*
									Pre	nvidia-smi (for b NVLink status c

11	**Throttle Reasons**	?		No equivalent	No equivalent	No		No equivalent *Redfish doesn't report specific throttle reasons such as whether the GPU is throttling due to power limits, thermal constraints, or being idle.* *It does not provide real-time data on GPU clock throttling activity.* *It does not differentiate between throttling due to software power caps or hardware issues.* *GET /redfish/v1/Chassis/{chassis_id}/Thermal* *GET /redfish/v1/Systems/{system_id}/Processors/{processor_id}* *GET /redfish/v1/Chassis/{chassis_id}/Power*	Pre and Post	nvidia-smi to ch reasons for thro

nvidia-smi --query-gpu=clocks_throttle_reasons.*

```
clocks_throttle_reasons.supported, clocks_throttle_reasons.active, clocks_throttle_reasons.gpu_idle, clocks_throttle_reasons.sw_power_cap, clocks_throttle_reasons.hw_slowdown, clocks_throttle_reasons.thermal_slowdown, clocks_throttle_reasons.power_limit
1, 0, 0, 1, 0, 0, 1
```

options for query-gpu: clocks_throttle_reasons.gpu_idle, clocks_throttle_reasons.applications_clocks_setting, clocks_throttle_reasons.sw_power_cap, clocks_throttle_reasons.hw_slowdown, clocks_throttle_reasons.thermal_slowdown, clocks_throttle_reasons.power_limit

can use --format=csv

12	**Driver Version**	?	nvidia-smi --query-gpu=driver_version	No equivalent	No equivalent	No		No equivalent	Pre	nvidia-smi (for c compatibility)

13	**Xid Errors (GPU Crashes)**	?	nvidia-smi --query-gpu=index, xid --format=csv
			`index, xid`
			`0,0`
			`1,3`
			`2,0`
			nvidia-smi --query-hwmon=pid,xid --format=csv
			`pid, xid`
			`1234,3`
			nvidia-smi -q -d ERRORS > nvidia_errors.log
				No equivalent	No equivalent	No equivalent	journalctl -k. \ dmesg grep -i xid	No equivalent	Pre and Post	

14	**Telemetry Collection**	yes	No equivalent
			nvidia-smi --query-gpu=temperature.gpu,utilization.gpu,memory.used,memory.total --format=csv --loop=5
			`temperature.gpu, utilization.gpu, memory.used, memory.total`
			`60, 75 %, 2048 MiB, 8192 MiB`
			`60, 76 %, 2050 MiB, 8192 MiB`
			the loop i to repeat every X seconds.
				No equivalent	dcgmi telemetry --interval <seconds>
					***Offers comprehensive telemetry data, including various metrics in one command, optimized for NVIDIA GPUs in data center environments.*
						Yes		GET /redfish/v1/Systems/<system_id>/GPU/<gpu_id>/Telemetry
								***The granularity and the specific metrics available can vary significantly based on the implementation. Some metrics available in `nvidia-smi` and `dcgmi` may not be exposed through Redfish.*
									Pre and Post	dcgmi for contin telemetry data c

15	**Remote GPU Management**	?	No equivalent
			
			***nvidia-smi** can perform GPU management commands but requires remote access (e.g., via SSH) to execute commands on the target machine.*
				No equivalent
				
				*nvvs can be run on remote systems via tools like **SSH**, similar to `nvidia-smi`.*
					dcgmi --list-gpus,
					
					dcgmi status --all
					
					`dcgmi health`
					
					`dcgmi setpower`
					
					`dcgmi utilization`
					
					`dcgmi telemetry`
					
					`dcgmi health` to check the health status of GPUs remotely.
					
					`dcgmi setpower` allow for remote control of power caps for individual GPUs or all GPUs in a data center.
					
					`dcgmi utilization`, you can monitor the utilization of GPUs across multiple nodes from a remote system.
					
					`dcgmi telemetry` command provides detailed logs of GPU performance and telemetry data, useful for long-term monitoring and management of remote GPUs.
						Yes		GET /redfish/v1/Chassis/{chassis_id}/Thermal
								
								GET /redfish/v1/Chassis/{chassis_id}/Power
								
								GET /redfish/v1/Systems/{system_id}/Processors
								
								GET /redfish/v1/Systems/{system_id}/LogServices
								
								GET /redfish/v1/Systems/{system_id}/Actions/Oem/NVIDIA/Reset
								
								GET /redfish/v1/Systems/{system_id}/Metrics
								
								*RedFish API can be used for **remote GPU management**, though it primarily offers high-level system monitoring and management, including power, temperature, and error logs.*
									Pre and Post	dcgmi for large-GPU managem

16	**Power Cap Management**	?	nvidia-smi --query-gpu=power.limit	No equivalent	dcgmi setpower --gpu <id> --limit <wattage>	Yes		GET /Chassis/<chassis-id>/Power GET /redfish/v1/Systems/{system_id}/Power	Pre and Post	dcgmi to control limits remotely
17	**Detailed GPU Utilization Across Nodes**	?	nvidia-smi --query-gpu=utilization.* *Full GPU utilization data: core, memory, encoder, decoder. good choice for moitoring GPU utilization*	No equivalent *Focused on diagnostics and validation, but does not provide real-time GPU utilization metrics* *Missing real-time utilization metrics like GPU core utilization, memory usage, and encoder/decoder utilization*	dcgmi utilization --gpu	Yes		No equivalent ***Specific GPU utilization metrics** (e.g., GPU, memory, and encoder/decoder utilization) are not fully exposed in most Redfish API implementations.*	Pre and Post	dcgmi for utiliza multi-GPU syste
18	**Systemwide GPU Stress Test**	yes	No equivalent *`nvidia-smi` tool does not have a built-in stress-testing feature*	nvvs -t stress.	dcgmi stress --gpu dcmg stress --gpu <id>	Yes		No equivalent	Post	NVVS or dcgmi systemwide stre validation

19	**PCIe Bandwidth and Utilization**	yes	nvidia-smi pcie	No equivalent	dcgmi diag -g 1	Yes		No equivalent	Pre and Post	nvidia-smi for ba dcgmi for deepe diagnostics

20	**Fan Speed and Fan Health**	yes	nvidia-smi --query-gpu=fan.speed
			`# providesfan speed as a percentage of the maximum possible speed that the GPU fan can achieve.`
			`fan.speed [%]`
			`40%`
				No equivalent	No equivalent	Yes		GET /Chassis/<chassis-id>/Thermal
								I don't believe, Redfish API does not natively expose the GPU's individual fan speeds in most implementations. The fan-related data is more general, covering system cooling, but not GPU-specific hardware.
									Pre and Post	nvidia-smi (for b
										fan speed monit

21	**Core and Memory Clock Speeds**	yes	nvidia-smi --query-gpu=clocks.gr,clocks.mem	nvvs -t clocks	dcgmi stats --pid	Yes		No equivalent	Pre and Post	nvidia-smi for re dcgmi for long-t analysis

Appendix

Appendix A) references

- WIP: Repair Management Probes
- Repair Management Scope for Network Link/Switch repair and firmware upgrades
- Repair Agent - Design
- NVIDIA GB200
- NVIDIA DCGM Documentation
- GB200 Hardware architecture and components
- nvidia-smi docs
- NVRASTool Test Setup (from the Red team)
- Redfish Spec
    - https://docs.nvidia.com/dgx/dgxh100-user-guide/redfish-api-supp.html
    - https://docs.nvidia.com/dgx/dgxa100-user-guide/redfish-api-supp.html
- Redfish API for NVIDIA
- Redfish API for BMC, Firmware updates
- NVOS_NVL_v25_02_1638_User_Manual.pdf
- ILOM Redfish API for Nvidia HMC [G4_8C]
- NVIDIA_Data_Center_Products_Telemetry_Catalog_v3.2 (1).xlsx. (This is an attachment)
- Telemetry Catalog: Telemetry Catalog.xlsx
- *Partnerdiag Tool references: DU-11965-001_04.pdf*
- GB200 NVL Manufacturing and Field Diags.pdf. GB200 NVL. Manufacturing and Field Diagnostics (part-2)
- *Testing hosts, SUNVTS sunvts_802-5331.pdf.*
- *GB200 NVL service flow user guid: NVIDIA GB200 NVL Service Flow User Guide.pdf*
- *nvidia debug tool user guid: NVOS_NVL_v25_02_1638_User_Manual.pdf* ***Also used for NVSwitch with NVOS REST APIs***

- GB200 SEA5 Rack 0116 Bring-Up Notes
- *Prototypes*
    - *https://bitbucket.oci.oraclecorp.com/users/klashgar/repos/probes_tool/browse*

Appendix B) Diagnostics tools

- **nvidia-smi** provides direct, granular data about GPU performance, temperature, memory, and more. `nvidia-smi` is commonly used for debugging and monitoring purposes on NVIDIA GPUs. `nvidia-smi` covers all critical aspects of GPU monitoring, including temperature, power consumption, memory, and utilization. This makes it suitable for detecting a wide range of issues such as overheating, under-utilization, or excessive memory usage.
- dcmg ca(Data Center GPU manager) provides monitoring and managing GPUs in a data center or server environment. **`dcmg`** command is designed for environments where multiple GPUs are used in servers. It offers more advanced features, especially in multi-GPU systems.

Appendix C) What is an Xid Message, error codes and possible causes

The Xid message is an error report from the NVIDIA driver that is printed to the operating system's kernel log or event log. Xid messages indicate that a general GPU error occurred, most often due to the driver programming the GPU incorrectly or to corruption of the commands sent to the GPU. The messages can be indicative of a hardware problem, an NVIDIA software problem, or a user application problem. These messages provide diagnostic information that can be used by both users and NVIDIA to aid in debugging reported problems.

Here are a list of all the Xid error listings along with their potential causes.

Under Linux, the Xid error messages are placed in the location `/var/log/messages`. Grep for "**NVRM: Xid**"to find all the Xid messages.

```
dmesg | grep -i nvidia
[...] NVRM: Xid (PCI:0000:02:00): 79, GPU has fallen off the bus
[...] CPU: 4 PID: 1234 Comm: nvidia-smi Tainted: P OE 4.15.0-147-generic
```

We can use Redfish APIU to get XIDs

```
 curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_0/LogServices/XID
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/LogServices/XID",
  "@odata.type": "#LogService.v1_1_0.LogService",
  "DateTime": "2025-01-08T14:23:16+00:00",
  "DateTimeLocalOffset": "+00:00",
  "Description": "XID Log Service",
  "Entries": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/LogServices/XID/Entries"
  },
  "Id": "XID",
  "Name": "XID Log Service",
  "Oem": {
    "Nvidia": {
      "@odata.type": "#NvidiaLogService.v1_3_0.NvidiaLogService",
      "BootEntryID": "184387",
      "LatestEntryID": "0",
      "LatestEntryTimeStamp": "1970-01-01T00:00:00+00:00"
    }
  },
  "OverWritePolicy": "WrapsWhenFull"
}[(flash)root@ORACLESP-1332524050198:~]#
```

Xid 1: GPU Hang / Watchdog Timeout: This error occurs when the GPU is no longer responding to commands, often referred to as a GPU "hang." It can be caused by long-running GPU tasks exceeding timeouts, driver issues, or hardware problems. some possible causes are

- Overloaded GPU tasks exceeding driver timeouts.
- Hardware failure or instability.
- Poor cooling or overheating.

Xid 6: PBDMA Error (DMA Engine Error): This error indicates an issue with the **PBDMA (Push Buffer DMA)** engine, which is responsible for transferring data between the host and the GPU. Some possible causes:

- Memory corruption or failures.
- Faulty PCIe communication between the CPU and GPU.

Xid 13: GR Engine Error (Graphics Engine Error): This points to an issue with the GPU's **Graphics Engine**, often due to corrupted instructions being sent to the GPU. Some possible causes:

- Faulty software or driver bugs.
- Hardware issues, including memory corruption or unstable overclocking.

Xid 31: VRAM Corruption Detected: The GPU's Video RAM (VRAM) has detected corruption or inconsistencies. Some possible causes:

- Failing memory modules.
- Overclocked memory causing instability.
- Driver or firmware bugs related to memory management.
- May indicate an NVLink error or problem with communication between GPUs.

Xid 32: Page Fault: The GPU encountered an invalid memory access (similar to a CPU page fault). Some possible causes:

- Memory corruption.
- Insufficient power to the GPU.
- Driver or firmware bugs.

Xid 43: GPU Exception / Graphics Engine Exception: A critical exception occurred in the GPU, often in the context of 3D rendering or compute tasks. Some possible causes:

- Overloading the GPU with complex tasks.
- Driver/firmware bugs.
- Hardware instability or failure.

Xid 45: Memory Scrubber Error: This error relates to the GPU's **memory scrubber**, a component that ensures memory integrity. The **GPU memory scrubber** is a feature found in some NVIDIA GPUs that helps to maintain the integrity of the GPU's memory. It works by periodically scanning the GPU's memory for errors and correcting them if necessary. Some possible causes:

- Hardware issues in the memory subsystem.
- Potential VRAM failures or ECC errors.

Xid 48: Power Supply Error / Thermal Event: This error indicates that the GPU detected a power failure or a thermal issue (such as overheating). Some possible causes:

- Insufficient or unstable power supply to the GPU.
- Cooling issues, leading to overheating.

Xid 56: Driver/Kernel Issue: This error occurs when the driver or kernel encounters an unexpected issue in its operations. It may be related to software, rather than hardware. Some possible causes:

- Driver bugs or conflicts.
- Incompatible software or OS updates.

Xid 61: Host/PCIe Error: The GPU encountered an issue when communicating over the **PCIe bus** with the host (CPU/motherboard). Some possible causes:

- PCIe bandwidth issues.
- Faulty PCIe slot, cable, or power supply.

Xid 62 Internal micro-controller halt (newer drivers): could point to memory errors or NVLink link issues. Some possible causes:

- HW errors
- Driver Errors
- Thermal Issues, (may relate to NVLink issues)

Xid 69: Illegal Instruction Error: The GPU received an illegal instruction that it could not process, possibly due to corruption in the command buffer. Some possible causes:

- Software bugs.
- Faulty hardware.

Xid 79: Context Switch Timeout: The GPU took too long to switch between contexts (e.g., tasks), leading to a timeout. Some possible causes:

- Overloaded GPU or long-running tasks.
- Driver bugs or kernel-related issues.

Xid 80: GPU Halt: The GPU has stopped processing commands and halted. Some possible causes:

- GPU power failure or hardware malfunction.
- Driver failure.

Xid 81: GPU Thermals / Fan Speed Issues: This error indicates that the GPU experienced temperature issues or fan speed control problems. some possible causes:

- Overheating due to cooling failure.
- Thermal paste degradation or dust accumulation

Xid 99: Unknown Error: An undefined or unknown error occurred. possible causes:

- Hardware or software malfunction, possibly needing deeper investigation. Usually hard to tell, probably best to remove the device and send to manufacturer for deeper diagnoses.

Xid 145 Packet Loss.

- **User Channels are RC'ed. CUDA calls return errors**
- **Xid Visible via compute BMC, NVML, DCGM, Kernel logs**
- **System Software automatically recovers ibn most cases**
- **Node reboot required to recove r in some cases. INdicated via NVML API.**

  Node with failed access Link:

  - **reset GPU indicated by Xid 154.**
    - Repieat access link failures indicate permanent failure based on some heuristic.
    - If permanent failure is indicated, run field diags.
  - All compute nodes:
    - Check GPU fault recovery action needed via NVML API, Xid 154
      - No action needed in most cases
      - Node reboot required in some cases.
    - Check fabric state via NVML API
      - Should be healthy
    - Restart app from previous checkpoint.

Xid 149: indicates link down on (compute tray)

Fabiric Side Visibility:

- NMX-C, NVOS CLI, gNMI: Port up/down
- NMX-T, gNMI (N/A) , NVOS CLI (POST TTM), REST API (POST TTM)
  - Phy counters (e.g. Link down info: LOCAL0REASON-OPCODE/REMOTE-REASON-OPCODE, PORT-RCV-ERRORSPORT-XMIT-DISCARDS)

Appendix D) Additional options on Temperature Monitoring

We can query our system's **SDR (Sensor Data Records)**, which should include various sensor readings such as temperatures, voltages, fan speeds, and more. **ipmitool** is another mechanism to investigate overheating on the liquid cooled hardware. The status of the sensor reading from the ipmitool's output (e.g., ok, warning, or critical) tells us whether the temperature is within normal limits, approaching dangerous levels, or if immediate action is needed to prevent system damage. NVIDIA GB200 series GPUs should have Baseboard Management Controllers (BMCs). BMC is a dedicated microcontroller that provides remote management capabilities for a server.
We may be able to query **cooling sensors** related to the GPU’s liquid cooling system through IPMI. The output will display all temperature-related sensors on the system. The exact output will vary depending on our hardware (e.g., motherboard, cooling system, installed sensors).

```
# ipmitool can be found on the nvdebug package
ipmitool sdr list | grep -i temp

CPU Temp          | 68 degrees C      | ok
System Temp       | 45 degrees C      | ok
Inlet Temp        | 32 degrees C      | ok
Exhaust Temp      | 55 degrees C      | ok
DIMM Temp         | 70 degrees C      | ok
GPU1 Temp         | 65 degrees C      | ok
GPU2 Temp         | 67 degrees C      | ok

The third column indicates sensor status.

to call the command remotely:
ipmitool -I lanplus -H <IPMI_IP_ADDRESS> -U <USERNAME> -P <PASSWORD> sdr list | grep -i temp

nvidia-imex-ctl -N
```

Appendix E) NVLink

NVLink is a high-speed, energy-efficient interconnect technology developed by NVIDIA to facilitate faster and more efficient communication between GPUs (Graphics Processing Units) and between GPUs and CPUs (Central Processing Units). Introduced as an advancement over traditional interconnects like PCI Express (PCIe), NVLink aims to overcome the bandwidth and scalability limitations inherent in older technologies, thereby enhancing the performance of high-performance computing (HPC), artificial intelligence (AI), deep learning, and data-intensive applications.

NVLink allows GPUs to communicate directly with each other without going through the CPU or system memory. This is particularly useful in multi-GPU setups like NVIDIA DGX systems, where NVLink allows fast data sharing across GPUs for parallel computations.

Check the NVLink error counters for various links in the system. This makes it easy for clients to catch abnormalities and watch the health of the communication over nvlink. ALso, overheating or overloading can cause NVLink instability and eventual failure.

These test also should include PRBS Pseudo-random bit sequence) tests to validate integrety and link stability of the network interface. PRBS test send pseudo-random bit patterns across the NVLink lanes and check for errors to evaluate the reliability of the communicaiton link.

THe 'nvvs -t stress --json' includes stressing the NVLink interface amongs other components like GPU, memory etc.. This stress test incldues aspects like link reliability, data integrity, and error detection.

sample output

```
nvvs -t stress  --json

nvidia-smi nvlink --status
nvidia-smi nvlink --query

dcgmi nvlink --status

dcgmi nvlink --errors -g 0

+-----------------------------------+
| GPU ID: 0 | NVLINK Error Counts                            |
+-------------------------------------+
|Link 0     | CRC FLIT Error      | 0                         |
|Link 0     | CRC Data Error      | 0                         |
|Link 0     | Replay Error        | 0                         |
|Link 0     | Recovery Error      | 0                         |
|Link 1     | CRC FLIT Error      | 0                         |
|Link 1     | CRC Data Error      | 0                         |
|Link 1     | Replay Error        | 0                         |
|Link 1     | Recovery Error      | 0                         |
|Link 2     | CRC FLIT Error      | 0                         |
|Link 2     | CRC Data Error      | 0                         |
|Link 2     | Replay Error        | 0                         |
|Link 2     | Recovery Error      | 0                         |
|Link 3     | CRC FLIT Error      | 0                         |
|Link 3     | CRC Data Error      | 0                         |
|Link 3     | Replay Error        | 0                         |
|Link 3     | Recovery Error      | 0                         |
+-----------------------------------------+

# All error counts should be zero in a healthy system.

# ways to look for NVLink Errors:
dmesg | grep -i nvlink
dmesg | grep -i xid    <== xid errors could have NVLink errors.

journalctl -k | grep -i nvlink
journalctl -k | grep -i xid
```

GB200 Modes of failure and actions to take

gNMI events will be generated for all system health related events

the NVOS commands to verify NVLink switch health & health history per the user manual guidelines. NVOS NVL User Manual Section 4.2.3.13

```
nv show system health
nv show system health history



# we can also use the REST APIs as defined in NVOS_NVL_v25_02_1638_User_Manual.pdf

GET https://<ip>/nvue_v1/ib/device
GET https://<ip>/nvue_v1/ib/device/{device-id}

GET https://<ip>/nvue_v1/interface/{interface-id}/link/state
GET https://<ip>/nvue_v1/interface/{interface-id}/link/counters
GET https://<ip>/nvue_v1/interface/{interface-id}/link
```

1) Access NVLink Failure:

Impact: App will terminate, can be restarted from last checkpoint •

Recovery: GPU will lose NVLink P2P Support. NVML/Nvidia-smi will report whether recovery action like GPU reset/Compute Node reboot etc. is needed.

Failure Attribution:

- Tenant VM – Xid from Driver, NVML, Nvidia-SMI Link State as InActive, DCGM will report same information.
- Compute Tray BMC – NVLink State InActive, XID
- Switch Tray – Corresponding Switch Port in NMX-T/NVOS Telemetry will report port state, error & stats information.
- NMX-C API – NVLink state up/down

2) Trunk Link Failure:

Impact: App will terminate, can be restarted from last checkpoint

Recovery: • NVML/Nvidia-smi will report whether recovery action like GPU reset/Compute Node reboot etc. is needed

NVML/Nvidia-smi will report whether BW is degraded, or GPU is not part of the Fabric

Recovery Options: Adaptive BW Mode: Drop the link – Keep all the compute, but run with less bandwidth across entire NVLink domain/partition Full BW Mode: Lose compute. Disable #GPUs proportional to bandwidth loss. Maintain Full BW for rest of NVLink domain

Failure Attribution:

- Tenant VM – Xid from Driver, NVML, Nvidia-SMI, DCGM will report same information
- Compute Tray BMC – Report XID,
- Switch Tray – Corresponding Switch Port in NMX-T/NVOS Telemetry will report port state, error & stats information
- NMX-C API – NVLink state up/down

3) Compute Tray Failure:

Impact: App will terminate, can be restarted from last checkpoint

Recovery: NVLink Domain can operate with less compute tray. Check for non-GPU errors • NVML/Nvidia-smi will report whether recovery action like GPU reset/Compute Node reboot etc. is needed

Failure Attribution:

- Tenant VM – Xid from Driver, NVML, Nvidia-SMI, DCGM will report same information
- Compute Tray BMC – Reports which component failed (SEL Logs)
- Switch Tray – Corresponding Switch Port in NMX-T/NVOS Telemetry will report port state, error & stats information (if GPU lost power/Link went down)
- NMX-C API – NVLink state up/down, GPU/node state

4) Switch Tray Failure:

Impact: App will terminate, can be restarted from last checkpoint

Recovery: Schedule cabinet level maintenance window

Failure Attribution:

(Assuming Switch ASIC Level Failure, Tray booted)

- Tenant VM: – Xid from Driver (if failed during workload) NVML, Nvidia-SMI report GPU Link # State as InActive, DCGM will report same information.
- Compute Tray BMC – GPU NVLink State as InActive
- Switch Tray – NVOS/gNMI fatal error report
- NMX-C API – NVLink neighbor state change, switch node state change

Evaluating the overall health and functionality of an NVSwitch in a given GB200 rack system

For the ROMA personnel to evaluate whether an NVSwitch in the NVIDIA GB200 rack is functioning properly, we can following diagnostic steps:

Check NVLink Connectivity: Use a GPU server to check the status of NVLink connections between GPUs. All NVLink connections should be reported as "Up" to ensure the NVSwitch is functioning correctly. If any links are down or degraded, it could indicate a problem with the NVSwitch.

*****If we see XID 149 three times in 72 hour window, then the NVLink needs to be replaced.***

```
nvidia-smi nvlink --status
```

Examine Fabric Manager Logs: On the host system, review the NVIDIA Fabric Manager logs for any errors or warnings related to the NVSwitch or NVLink connections. Look for messages about failed initialization, degraded performance, or communication errors.

```
sudo journalctl -u nvidia-fabricmanager
```

Verify Topology: On a GPU server, view the GPU and NVSwitch topology to confirm that the GPUs are properly connected through the NVSwitch fabric. This will help ensure the inter-GPU communication routes are as expected.

```
nvidia-smi topo -m
```

Monitor NVLink Performance: Track NVLink usage and bandwidth on the GPU server. Consistent or high NVLink usage indicates that the NVSwitch is performing as expected. Any drop in performance could signal issues with the NVSwitch or NVLink fabric.

```
nvidia-smi topo -m
```

Check Kernel Logs: Review system logs on the GPU server to look for any hardware-related messages or errors related to the NVSwitch or NVLink, such as GPU or NVIDIA errors.

```
dmesg | grep NVSwitch
```

Use Monitoring Tools: Utilize tools like NVIDIA’s Data Center GPU Manager (DCGM) for continuous monitoring of NVLink and NVSwitch health. This can help automate the detection of NVSwitch failures or performance degradation.

```
dmesg | grep -i nvidia
```

Inspect Hardware Indicators: If available, check the physical NVSwitch for any error indicators, such as LEDs, that might suggest hardware faults or connection issues.

```
dcgmi nvlink --status
```

Appendix E) Running commnads remotely using SSH

Stress tesing remotely

To run the commands like "***`nvvs -t stress`***" remotely on a GB200 system via SSH, we'll need to ensure that we are targeting the correct component (usually the GPU server or node) where the NVIDIA GPUs are installed and managed.

i. **Identify the Target System/Component**: The GB200 is a sled system and each sled typically hosts one or more GPUs, which can be managed remotely. When connecting to a GB200 sled we first need to identify the node or server that houses the GB200 slet. Once we have identified the target node or server that houses the GB200 sled, we need to SSH into that the system hosting the GPU (e.g. ssh user@<remote_server>).

ii. **Ensure Prerequisites are met**: We must ensure that the **NVVS (NVIDIA Validation Suite)** and/or **DCGM (Data Center GPU Manager)** tools are installed (e.g. run nvvs --version)

iii. **Run the stress test:** We must run the stress test and collect its output into a file. (e.g. run. *nvvs -t stress --output nvvs_stress_results.json.*). The resulting test output can be either pushed to OCI Object Store or thorugh the '*scp user@<remote_server>:/path/to/nvvs_stress_results.json /local/path*' command sent back to the test orchestration host for further analysis before pushing to safe location.
iv. **Exit out of the system**: close the ssh session.

Appendix F) nvidia-smi example:

```
root@gb200-dvt-0:~# nvidia-smi --query-gpu=timestamp,pci.bus_id,clocks_throttle_reasons.sw_power_cap,
clocks_throttle_reasons.hw_slowdown,clocks_throttle_reasons.hw_power_brake_slowdown,clocks_throttle_reasons.
sw_thermal_slowdown,temperature.gpu,power.draw -l 1 --format=csv
timestamp, pci.bus_id, clocks_event_reasons.sw_power_cap, clocks_event_reasons.hw_slowdown,
clocks_event_reasons.hw_power_brake_slowdown, clocks_event_reasons.sw_thermal_slowdown, temperature.gpu, power.
draw [W]
2024/12/12 00:24:42.254, 00000019:01:00.0, Not Active, Not Active, Not Active, Not Active, 49, 518.42 W
2024/12/12 00:24:43.269, 00000008:01:00.0, Not Active, Not Active, Not Active, Not Active, 47, 523.85 W
2024/12/12 00:24:43.284, 00000009:01:00.0, Not Active, Not Active, Not Active, Not Active, 48, 481.50 W
2024/12/12 00:24:43.299, 00000018:01:00.0, Not Active, Not Active, Not Active, Not Active, 49, 486.49 W
2024/12/12 00:24:43.317, 00000019:01:00.0, Not Active, Not Active, Not Active, Not Active, 49, 517.58 W

root@gb200-dvt-0:~# nvidia-smi
Thu Dec 12 00:22:29 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 570.26 Driver Version: 570.26 CUDA Version: 12.8 |
|-----------------------------------------+------------------------+----------------------+
| GPU Name Persistence-M | Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap | Memory-Usage | GPU-Util Compute M. |
| | | MIG M. |
|=========================================+========================+======================|
| 0 NVIDIA Graphics Device On | 00000008:01:00.0 Off | 0 |
| N/A 39C P0 215W / 1000W | 169401MiB / 189471MiB | 1% Default |
| | | Disabled |
+-----------------------------------------+------------------------+----------------------+
| 1 NVIDIA Graphics Device On | 00000009:01:00.0 Off | 0 |
| N/A 39C P0 185W / 1000W | 169465MiB / 189471MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+------------------------+----------------------+
| 2 NVIDIA Graphics Device On | 00000018:01:00.0 Off | 0 |
| N/A 40C P0 190W / 1000W | 169465MiB / 189471MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+------------------------+----------------------+
| 3 NVIDIA Graphics Device On | 00000019:01:00.0 Off | 0 |
| N/A 40C P0 233W / 1000W | 169465MiB / 189471MiB | 4% Default |
| | | Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes: |
| GPU GI CI PID Type Process name GPU Memory |
| ID ID Usage |
|=========================================================================================|
| 0 N/A N/A 51224 C ./nvgputest 16938... |
| 1 N/A N/A 51224 C ./nvgputest 16945... |
| 2 N/A N/A 51224 C ./nvgputest 16945... |
| 3 N/A N/A 51224 C ./nvgputest 16945... |
+-----------------------------------------------------------------------------------------+
```

Appendix G) NVQUEL provides a list of the tests, descriptions, and estimated test duration.

The following table will describe the test description requried to return a unit back to NVIDIA.

examples:

```
# To run CX7 PCIe Test (test 16), run the following command.
./nvqual --bypass_menu --tests 16

# To run the Thermal Test (test 1) with the SSD subtest, the CX7 PCIe Test (test 16), and
# install the NVQUAL dependencies, run the following command.
./nvqual --bypass_menu --tests 1 1.2 16
```

**NVIDIA document reference: *NVIDIA Qualification User Guide for NVIDIA GB200 NVL.pdf*

Test Number	Test Name	Description	Estimated Test Duration	Requirements
1	System Thermal Qualificaiton Test	The thermal qualification Test is recommended for system thermal qualifications. The test reuns on all cpuS AND GPUS that are installed in a system and consume a power level that is equal to the rated TGP of the product. Refer to the product specification guide for the rated TGP of the product. The Thermal Qualification test can also stress CX7, BF3, and the NVME SSDs that are available on the platform.	1 hr (all CPUs, GPUs, NVME SSDs, CX7s)	SSD Stress: LINUX FIO LINUX IOSTAT Bluefield-3 Stress: OFED DOCA Latest BF3 OS External Loopback Cable ConnectX-7 Stress: OFED

Appendix H) Example Redfish API calls and outputs.

getting Serial Numbers for a GB200 GPU server from a ILOM using sunservice

```
[klashgar@pxe-server ~]$ ssh sunservice@100.72.0.67


[(flash)root@ORACLESP-2504XKG006:~]#redfishclient get /redfish/v1/Chassis/HGX_BMC_0 | grep SerialNumber
[(flash)root@ORACLESP-2504XKG006:~]#redfishclient get /redfish/v1/Chassis/HGX_CPLD_0 | grep SerialNumber
[(flash)root@ORACLESP-2504XKG006:~]#redfishclient get /redfish/v1/Chassis/HGX_CPU_0 | grep SerialNumber
[(flash)root@ORACLESP-2504XKG006:~]#redfishclient get /redfish/v1/Chassis/HGX_CPU_1 | grep SerialNumber
    6  redfishclient get /redfish/v1/Chassis/HGX_Chassis_0 | grep SerialNumber
    7  redfishclient get /redfish/v1/Chassis/HGX_ERoT_BMC_0 | grep SerialNumber
    8  redfishclient get /redfish/v1/Chassis/HGX_ERoT_CPU_0 | grep SerialNumber
    9  redfishclient get /redfish/v1/Chassis/HGX_ERoT_CPU_1 | grep SerialNumber
   10  redfishclient get /redfish/v1/Chassis/HGX_ERoT_FPGA_0 | grep SerialNumber
   11  redfishclient get /redfish/v1/Chassis/HGX_ERoT_FPGA_1 | grep SerialNumber
   12  redfishclient get /redfish/v1/Chassis/HGX_FPGA_0 | grep SerialNumber
   13  redfishclient get /redfish/v1/Chassis/HGX_FPGA_1 | grep SerialNumber
   14  redfishclient get /redfish/v1/Chassis/HGX_GPU_0 | grep SerialNumber
   15  redfishclient get /redfish/v1/Chassis/HGX_GPU_1 | grep SerialNumber
   16  redfishclient get /redfish/v1/Chassis/HGX_GPU_2 | grep SerialNumber
   17  redfishclient get /redfish/v1/Chassis/HGX_GPU_3 | grep SerialNumber
   18  redfishclient get /redfish/v1/Chassis/HGX_GPU_4 | grep SerialNumber
   19  redfishclient get /redfish/v1/Chassis/HGX_ProcessorModule_0 | grep SerialNumber
   20  redfishclient get /redfish/v1/Chassis/HGX_ProcessorModule_1 | grep SerialNumber
```

These calls are all doen on the GB200.

```
This NVIDIA doc has some good Redfish examples: NVIDIA_GB200_NVL Redfish_Bundle_QS_v1.1

}[(flash)root@ORACLESP-1332524050198:~]# curl http://172.31.13.251/redfish/v1
/Chassis
{
  "@odata.id": "/redfish/v1/Chassis",
  "@odata.type": "#ChassisCollection.ChassisCollection",
  "Members": [
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_BMC_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_CPLD_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_CPU_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_CPU_1"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_Chassis_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_ERoT_BMC_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_ERoT_CPU_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_ERoT_CPU_1"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_ERoT_FPGA_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_ERoT_FPGA_1"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_FPGA_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_FPGA_1"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_1"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_3"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_ProcessorModule_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_ProcessorModule_1"
    }
  ],
  "Members@odata.count": 18,
  "Name": "Chassis Collection"
[(flash)root@ORACLESP-1332524050198:~]#
```

```
[(flash)root@ORACLESP-1332524050198:~]# curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_0/PowerSubsystem
/PowerSupplies
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/PowerSubsystem/PowerSupplies",
  "@odata.type": "#PowerSupplyCollection.PowerSupplyCollection",
  "Description": "The collection of PowerSupply resource instances.",
  "Members": [],
  "Members@odata.count": 0,
  "Name": "Power Supply Collection"
}[(flash)root@ORACLESP-1332524050198:~]#

}[(flash)root@ORACLESP-1332524050198:~]# curl http://172.31.13.251/redfish/v1/Chassis/HGX_Chassis_0
/PowerSubsystem
{
  "@odata.id": "/redfish/v1/Chassis/HGX_Chassis_0/PowerSubsystem",
  "@odata.type": "#PowerSubsystem.v1_1_0.PowerSubsystem",
  "Id": "PowerSubsystem",
  "Name": "Power Subsystem",
  "PowerSupplies": {
    "@odata.id": "/redfish/v1/Chassis/HGX_Chassis_0/PowerSubsystem/PowerSupplies"
  },
  "Status": {
    "Health": "OK",
    "State": "Enabled"
  }




curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_2
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2",
  "@odata.type": "#Chassis.v1_22_0.Chassis",
  "Assembly": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Assembly"
  },
  "ChassisType": "Module",
  "Controls": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Controls"
  },
  "DepthMm": 0.0,
  "EnvironmentMetrics": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/EnvironmentMetrics"
  },
  "HeightMm": 0.0,
  "Id": "HGX_GPU_2",
  "Links": {
    "ComputerSystems": [
      {
        "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0"
      }
    ],
    "ContainedBy": {
      "@odata.id": "/redfish/v1/Chassis/HGX_Chassis_0"
    },
    "ManagedBy": [
      {
        "@odata.id": "/redfish/v1/Managers/HGX_BMC_0"
      }
    ],
    "Processors": [
      {
        "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2"
      }
    ]
  },
  "Location": {
    "PartLocation": {
      "LocationType": "Embedded"
    }
  },
  "LogServices": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/LogServices"
  },
  "Manufacturer": "NVIDIA",
  "MaxPowerWatts": 0,
  "MinPowerWatts": 0,
  "Model": "",
  "Name": "HGX_GPU_2",
  "Oem": {
    "Nvidia": {
      "@odata.type": "#NvidiaChassis.v1_1_0.NvidiaChassis"
    }
  },
  "PCIeDevices": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/PCIeDevices"
  },
  "PCIeSlots": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/PCIeSlots"
  },
  "PartNumber": "",
  "PowerSubsystem": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/PowerSubsystem"
  },
  "SKU": "",
  "Sensors": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors"
  },
  "SerialNumber": "",
  "Status": {
    "Conditions": [],
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  },
  "ThermalSubsystem": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/ThermalSubsystem"
  },
  "TrustedComponents": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/TrustedComponents"
  },
  "UUID": "$DEVICE_UUID",
  "WidthMm": 0.0
}[(flash)root@ORACLESP-1332524050198:~]#
```

```
[(flash)root@ORACLESP-1332524050198:~]# curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_2
/EnvironmentMetrics
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/EnvironmentMetrics",
  "@odata.type": "#EnvironmentMetrics.v1_3_0.EnvironmentMetrics",
  "Actions": {
    "Oem": {
      "Nvidia": {
        "#NvidiaEnvironmentMetrics.ClearOOBSetPoint": {
          "target": "/redfish/v1/Chassis/HGX_GPU_2/EnvironmentMetrics/Actions/Oem/NvidiaEnvironmentMetrics.
ClearOOBSetPoint"
        }
      }
    }
  },
  "EnergyJoules": {
    "DataSourceUri": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_Energy_0",
    "Reading": null
  },
  "EnergykWh": {
    "Reading": null
  },
  "FanSpeedsPercent": [],
  "Id": "EnvironmentMetrics",
  "Name": "Chassis Environment Metrics",
  "Oem": {
    "Nvidia": {
      "@odata.type": "#NvidiaEnvironmentMetrics.v1_0_0.NvidiaEnvironmentMetrics"
    }
  },
  "PowerWatts": {
    "DataSourceUri": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_Power_0",
    "Reading": null
  }
}[(flash)root@ORACLESP-1332524050198:~]#
```

```
 curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_2/Sensors
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors",
  "@odata.type": "#SensorCollection.SensorCollection",
  "Description": "Collection of Sensors for this Chassis",
  "Members": [
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_Energy_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_DRAM_0_Power_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_Power_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_DRAM_0_Temp_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_TEMP_0"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_TEMP_1"
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_Voltage_0"
    }
  ],
  "Members@odata.count": 7,
  "Name": "Sensors"
}[(flash)root@ORACLESP-1332524050198:~]#
```

```
curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_Power_0
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_Power_0",
  "@odata.type": "#Sensor.v1_2_0.Sensor",
  "Id": "HGX_GPU_2_Power_0",
  "MaxAllowableOperatingValue": 1020,
  "Name": "HGX GPU 2 Power 0",
  "PhysicalContext": "GPU",
  "Reading": null,
  "ReadingTime": "1970-01-01T00:00:00.000+00:00",
  "ReadingType": "Power",
  "ReadingUnits": "W",
  "RelatedItem": [
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2"
    }
  ],
  "Status": {
    "Conditions": [],
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  }
}[(flash)root@ORACLESP-1332524050198:~]#


 curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_TEMP_0
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_TEMP_0",
  "@odata.type": "#Sensor.v1_2_0.Sensor",
  "Id": "HGX_GPU_2_TEMP_0",
  "Name": "HGX GPU 2 TEMP 0",
  "PhysicalContext": "GPU",
  "Reading": null,
  "ReadingType": "Temperature",
  "ReadingUnits": "Cel",
  "RelatedItem": [
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2"
    }
  ],
  "Status": {
    "Conditions": [],
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  }
}[(flash)root@ORACLESP-1332524050198:~]#



curl http://172.31.13.251/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2
{
  "@Redfish.Settings": {
    "@odata.type": "#Settings.v1_3_3.Settings",
    "SettingsObject": {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2/Settings"
    }
  },
  "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2",
  "@odata.type": "#Processor.v1_20_0.Processor",
  "BaseSpeedMHz": 0,
  "EnvironmentMetrics": {
    "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2/EnvironmentMetrics"
  },
  "Id": "GPU_2",
  "Links": {
    "Chassis": {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2"
    },
    "Memory": [
      {
        "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Memory/GPU_2_DRAM_0"
      }
    ],
    "PCIeDevice": {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/PCIeDevices/GPU_2"
    },
    "PCIeFunctions": []
  },
  "Location": {
    "PartLocation": {
      "LocationType": "Embedded"
    }
  },
  "Manufacturer": "NVIDIA",
  "MaxSpeedMHz": 0,
  "MemorySummary": {
    "ECCModeEnabled": false,
    "Metrics": {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2/MemorySummary/MemoryMetrics"
    },
    "TotalCacheSizeMiB": 0,
    "TotalMemorySizeMiB": 0
  },
  "Metrics": {
    "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2/ProcessorMetrics"
  },
  "MinSpeedMHz": 0,
  "Name": "Processor",
  "Oem": {
    "Nvidia": {
      "@odata.type": "#NvidiaProcessor.v1_3_0.NvidiaGPU",
      "MIGModeEnabled": false,
      "SystemGUID": "$DEVICE_UUID"
    }
  },
  "OperatingSpeedMHz": 0,
  "PartNumber": "",
  "Ports": {
    "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2/Ports"
  },
  "ProcessorType": "GPU",
  "SpeedLimitMHz": 0,
  "SpeedLocked": false,
  "Status": {
    "Conditions": [],
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  },
  "SystemInterface": {
    "InterfaceType": "PCIe",
    "PCIe": {
      "LanesInUse": 4294967295,
      "MaxLanes": 0,
      "MaxPCIeType": "Unknown",
      "PCIeType": "Unknown"
    }
  },
  "UUID": "$DEVICE_UUID",
  "Version": ""
}[(flash)root@ORACLESP-1332524050198:~]#



curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_0/ThermalSubsystem
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/ThermalSubsystem",
  "@odata.type": "#ThermalSubsystem.v1_0_0.ThermalSubsystem",
  "Id": "ThermalSubsystem",
  "Name": "Thermal Subsystem",
  "Status": {
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  },
  "ThermalMetrics": {
    "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/ThermalSubsystem/ThermalMetrics"
  }

curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_0/ThermalSubsystem/ThermalMetrics
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/ThermalSubsystem/ThermalMetrics",
  "@odata.type": "#ThermalMetrics.v1_0_0.ThermalMetrics",
  "Id": "ThermalMetrics",
  "Name": "Chassis Thermal Metrics",
  "TemperatureReadingsCelsius": [
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/Sensors/HGX_GPU_0_TEMP_0",
      "DataSourceUri": "/redfish/v1/Chassis/HGX_GPU_0/Sensors/HGX_GPU_0_TEMP_0",
      "DeviceName": "HGX_GPU_0_TEMP_0",
      "PhysicalContext": "GPU",
      "Reading": null
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/Sensors/HGX_GPU_0_TEMP_1",
      "DataSourceUri": "/redfish/v1/Chassis/HGX_GPU_0/Sensors/HGX_GPU_0_TEMP_1",
      "DeviceName": "HGX_GPU_0_TEMP_1",
      "PhysicalContext": "GPU",
      "Reading": null
    },
    {
      "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/Sensors/HGX_GPU_0_DRAM_0_Temp_0",
      "DataSourceUri": "/redfish/v1/Chassis/HGX_GPU_0/Sensors/HGX_GPU_0_DRAM_0_Temp_0",
      "DeviceName": "HGX_GPU_0_DRAM_0_Temp_0",
      "PhysicalContext": "GPU",
      "Reading": null
    }
  ]
}[(flash)root@ORACLESP-1332524050198:~]#

curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_0/Sensors/HGX_GPU_0_TEMP_0
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_0/Sensors/HGX_GPU_0_TEMP_0",
  "@odata.type": "#Sensor.v1_2_0.Sensor",
  "Id": "HGX_GPU_0_TEMP_0",
  "Name": "HGX GPU 0 TEMP 0",
  "PhysicalContext": "GPU",
  "Reading": null,
  "ReadingType": "Temperature",
  "ReadingUnits": "Cel",
  "RelatedItem": [
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0"
    }
  ],
  "Status": {
    "Conditions": [],
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  }
}[(flash)root@ORACLESP-1332524050198:~]#


curl http://172.31.13.251/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_DRAM_0_Temp_0
{
  "@odata.id": "/redfish/v1/Chassis/HGX_GPU_2/Sensors/HGX_GPU_2_DRAM_0_Temp_0",
  "@odata.type": "#Sensor.v1_2_0.Sensor",
  "Id": "HGX_GPU_2_DRAM_0_Temp_0",
  "Name": "HGX GPU 2 DRAM 0 Temp 0",
  "PhysicalContext": "GPU",
  "Reading": null,
  "ReadingType": "Temperature",
  "ReadingUnits": "Cel",
  "RelatedItem": [
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_2"
    }
  ],
  "Status": {
    "Conditions": [],
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  },
  "Thresholds": {
    "UpperCritical": {
      "Reading": 95.0
    }
  }
}[(flash)root@ORACLESP-1332524050198:~]#
```

```
curl http://172.31.13.251/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_9#/LnkStatus
{
  "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_9",
  "@odata.type": "#Port.v1_4_0.Port",
  "CurrentSpeedGbps": 0.0,
  "Id": "NVLink_9",
  "LinkState": "Disabled",
  "LinkStatus": "Starting",
  "MaxSpeedGbps": 0.0,
  "Metrics": {
    "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_9/Metrics"
  },
  "Name": "NVLink_9 Resource",
  "Oem": {
    "Nvidia": {
      "@odata.type": "#NvidiaPort.v1_0_0.NvidiaPort",
      "RXWidth": 0,
      "TXWidth": 0
    }
  },
  "PortProtocol": "NVLink",
  "PortType": "BidirectionalPort",
  "Status": {
    "Conditions": [],
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  }
}[(flash)root@ORACLESP-1332524050198:~]#
```

```
[(flash)root@ORACLESP-1332524050198:~]# curl http://172.31.13.251/redfish/v1/Systems/HGX_Baseboard_0/Processors
/GPU_0/Ports/NVLink_9
{
  "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_9",
  "@odata.type": "#Port.v1_4_0.Port",
  "CurrentSpeedGbps": 0.0,
  "Id": "NVLink_9",
  "LinkState": "Disabled",
  "LinkStatus": "Starting",
  "MaxSpeedGbps": 0.0,
  "Metrics": {
    "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_9/Metrics"
  },
  "Name": "NVLink_9 Resource",
  "Oem": {
    "Nvidia": {
      "@odata.type": "#NvidiaPort.v1_0_0.NvidiaPort",
      "RXWidth": 0,
      "TXWidth": 0
    }
  },
  "PortProtocol": "NVLink",
  "PortType": "BidirectionalPort",
  "Status": {
    "Conditions": [],
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  }
}[(flash)root@ORACLESP-1332524050198:~]#




curl http://172.31.13.251/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports
{
  "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports",
  "@odata.type": "#PortCollection.PortCollection",
  "Members": [
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_0"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_1"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_2"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_3"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_4"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_5"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_6"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_7"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_8"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_9"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_10"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_11"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_12"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_13"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_14"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_15"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_16"
    },
    {
      "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Ports/NVLink_17"
    }
  ],
  "Members@odata.count": 18,
  "Name": "NVLink Port Collection"
}[(flash)root@ORACLESP-1332524050198:~]#
```

Appendix I) DCGMI

Fault injection using DCGMI

DCGM injection by this page? https://docs.nvidia.com/datacenter/dcgm/latest/user-guide/dcgm-error-injection.html

Additionally, I think a driver module removal can also be a valid method of simulating a GPU failure?

```
sudo rmmod nvidia_uvm nvidia_drm nvidia_modeset nvidia
```

We can blacklist the nvidia driver and reboot?, after reboot, the GPU will be failed to be loaded

```
echo "blacklist nvidia" | sudo tee -a /etc/modprobe.d/blacklist-nvidia.conf
sudo update-initramfs -u
```

Sample DCGMI output

resutls of the DCGM 4.0.0 on GB200 System:

```
# if you see the error -> Persistence Mode: Persistence mode for GPU 2 is disabled. Enable persistence mode by
running \"nvidia-smi -i <gpuId> -pm 1 \" as root.
sudo nvidia-smi -i 0,1,2,3 -pm 1
Enabled Legacy persistence mode for GPU 00000008:01:00.0.
Enabled Legacy persistence mode for GPU 00000009:01:00.0.
Enabled Legacy persistence mode for GPU 00000018:01:00.0.
Enabled Legacy persistence mode for GPU 00000019:01:00.0.



dcgmi diag -j -r 1
{
        "DCGM Diagnostic": {
            "test_categories": [
                {
                    "category": "Deployment",
                    "tests": [
                        {
                            "name": "software",
                            "results": [
                                {
                                    "entity_group": "GPU",
                                    "entity_group_id": 1,
                                    "entity_id": 0,
                                    "status": "Pass"
                                },
                                {
                                    "entity_group": "GPU",
                                    "entity_group_id": 1,
                                    "entity_id": 1,
                                    "status": "Pass"
                                },
                                {
                                    "entity_group": "GPU",
                                "entity_group_id": 1,
                                "entity_id": 2,
                                "status": "Pass"
                            },
                            {
                                "entity_group": "GPU",
                                "entity_group_id": 1,
                                "entity_id": 3,
                                "status": "Pass"
                            }
                        ],
                        "test_summary": {
                            "status": "Pass"
                        }
                    }
                ]
            }
        ]
    },
    "entity_groups": [
        {
            "entities": [
                {
                    "device_id": "2941",
                    "entity_id": 0,
                    "serial_num": "1643524000911"
                },
                {
                    "device_id": "2941",
                    "entity_id": 1,
                    "serial_num": "1643524000911"
                },
                {
                    "device_id": "2941",
                    "entity_id": 2,
                    "serial_num": "1643524000930"
                },
                {
                    "device_id": "2941",
                    "entity_id": 3,
                    "serial_num": "1643524000930"
                }
            ],
            "entity_group": "GPU",
            "entity_group_id": 1
        },
        {
            "entities": [
                {
                    "device_id": "",
                    "entity_id": 0,
                    "serial_num": "0x00000001780CA1092000000004008240"
                },
                {
                    "device_id": "",
                    "entity_id": 1,
                    "serial_num": "0x00000001780A01851400000007010200"
                }
            ],
            "entity_group": "CPU",
            "entity_group_id": 7
        }
    ],
    "metadata": {
        "Driver Version Detected": "570.82",
        "version": "4.0.0"
    }
}
```

Appendix J) partnerdiag

sample code for partnerdiag

GB200 Manual Run Outputs

For rack level do:partnerdiag-GB200-compute-tray-L10-629-24975-0000-FLD-42127.dms.tgz

- Compute tray diag is L10.
- Rack lecel diag is L11 .

```
sudo ./partnerdiag --field --level1 --run_on_error --no_bmc Either --primary_diag_ip or --gdm_fd needs to be
provided
./partnerdiag --field --level1

# install core
sudo dpkg -i datacenter-gpu-manager-4-core_4.0.0~10088_arm64.deb

# install cuda12 variant
sudo dpkg -i datacenter-gpu-manager-4-cuda12_4.0.0~10088_arm64.deb



Running partner diagnostics
  --------------------------
  1) cd <Diag Package Directory>
  2) Run Partner Manufacturing Diagnostics:
  - Mfg  :
    - Run on switch nodes
      ./partnerdiag --mfg --run_spec=spec_gb200_nvl_72_2_4_switch_nodes_partner_mfg.json --primary_diag_ip=<IP>
--topology=<NVLink topology json>
    - Run on compute nodes
      ./partnerdiag --mfg --run_spec=spec_gb200_nvl_72_2_4_compute_nodes_partner_mfg.json --
primary_diag_ip=<IP> --topology=<NVLink topology json>
  - Field :
    - Level 1
      ./partnerdiag --field --level1 --primary_diag_ip=<IP> --topology=<NVLink topology json>
    - Level 2
      ./partnerdiag --field --level2 --primary_diag_ip=<IP> --topology=<NVLink topology json>
  3) PASS/FAIL/RETEST will be displayed when partnerdiag finishes execution.
  4) Run ./partnerdiag --help for more details on options

  Useful Options
  --------------
  1) --skip_tests=<virtual_id>,<virtual_id> --> Skips the specified list of tests.
  2) --test=<virtual_id>,<virtual_id> --> Tests the specified list of tests.
```