RIPE NCC ACADEMY, IPV6 ADVANCED COURSE

In-Person Lab Exercise Booklet

Building AS-**U**00: IS-IS, BGP, and IPv6 L3VPN over SRv6

Student Guide

Last updated: September 2026

RIPE NCC Training

Your RIPE NCC Access Account

All services that the RIPE NCC offers are supported by the RIPE NCC Access Single Sign-On (SSO) system, including the hands-on lab environment used for this course (Workbench).

You must have a RIPE NCC Access account with Two-Factor Authentication (2FA) enabled to take this training course. If you do not have an account yet, create one in advance by following these instructions.

If you already have a RIPE NCC Access account, go to access.ripe.net and try to log in.
If you have any issues accessing your account, let the trainers know so they can try to help you.

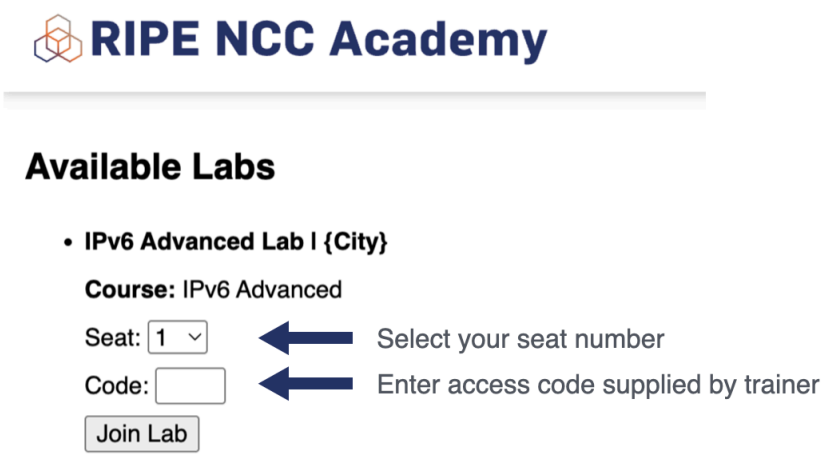
How to Access the Lab Environment

The trainer will assign you a number (Number **U**), which will be your **Seat** to access the lab environment. The trainer will also provide you with a **Code** to access the IPv6 Advanced lab.

Go to https://workbench.ripe.net and log in using your RIPE NCC Access account. If you don't have one, please register on this page.

After logging in, find the **"IPv6 Advanced"** lab, and then:

1. Select your **Seat** number from the dropdown menu
2. Enter the **Code** provided by the trainer.
3. Click on the **Join Lab** button.



You will see a landing page with the lab network diagram and tabs to access the lab nodes.

The routers in the lab network are running open-source routing software, FRR (Free Range Routing) version 10.5.0 (https://frrouting.org).

> ⚠ **Rebooting lab nodes**
>
> Do not try to reboot any device assigned to you. Rebooting will make them non-functional for the rest of the training.

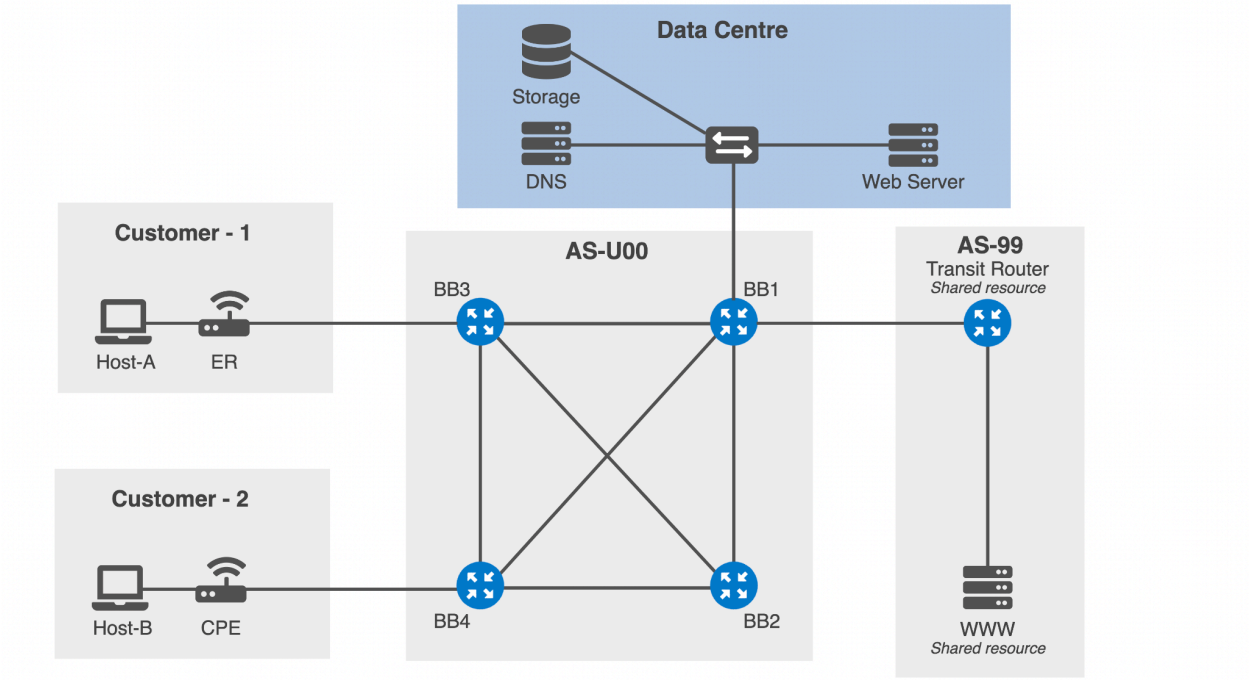
Introduction and Topology

You are the newest network engineer at ISP Sandbox Inc. (AS-**U**00), an IPv6-only operator that has just finished racking and cabling a small backbone. Two customers are already under contract and waiting for service: **Customer-1**, **who brings their Provider-Independent IPv6 block**, and **Customer-2, who uses address space delegated out of Sandbox Inc.'s own allocation**. Your job over the next five activities is to take this backbone from bare metal to two fully isolated, working IPv6 L3VPN services delivered over an SRv6 underlay, plus the addressing records, routing, and DNS that make it usable.

Lab technicians have physically cabled everything and pre-staged the devices that are not part of your learning objectives (see "Pre-Configured Elements" below). Everything else (**IS-IS**, **BGP**, **SRv6**, the **VRFs**, and **DNS**) is yours to build.

- This is a shared classroom lab: every seat gets its own fully isolated pod, its own BB1-BB4, its own Customer-1 and Customer-2 sites, and its own Data Centre (Storage, DNS, WEB), but every pod's BB1 peers into the same shared Transit-Router and simulated Internet (AS-99, WWW).
- Every address in this booklet that could otherwise collide with another seat's pod is written with the placeholder "**U**", which is your seat number as assigned by your trainer.
- Wherever you see "**U**" in a prefix or an AS number, replace it with your own seat number to get the real value for your pod. For example, if you are in seat 7, AS-**U**00 means AS-700, and 3fff:**U**00::1 means 3fff:700::1.
- VLAN IDs, interface names, and IS-IS system IDs are not seat-specific; every pod is internally identical (only the address space differs).
- Light-coloured command boxes show configurations that have to be inserted.
- Black-coloured Command boxes are for checking the current state of the lab nodes. Blue text is for commands you enter; Orange text is for the expected output.

Lab Network Overview



The lab consists of four backbone routers (BB1-BB4) forming AS-**U**00, a Data Centre behind a VLAN-aware switch with three servers (Storage, DNS, WEB), two customer sites (Customer-1 via the Enterprise Router ER, Customer-2 via router CPE), and a Transit-Router representing AS-99 and the wider IPv6 Internet (with a WWW server behind it).

- AS-**U**00 (BB1-BB4): IPv6-only SRv6 backbone, IS-IS Level-2, no IPv4 in the core.
- Customer-1: PI address space (2001:db8:**U**23::/48), connects at BB3 via ER.
- Customer-2: address space delegated from AS-**U**00 (3fff:**U**00:21::/48), connects at BB4 via CPE.
- Data Centre: Storage (Customer-1's private service), WEB (Customer-2's private service), and DNS (a shared service reachable by both customers).
- AS-99: reached via eBGP from BB1 to the Transit-Router; the WWW server simulates the public Internet.

Pre-Configured Elements (Do Not Touch)

Element	What's Already Done
All device interfaces	Every interface on every device already has its final IPv6 address configured. You are not addressing anything, only enabling protocols.
Enterprise-Router (ER)	Fully pre-configured: WAN link to BB3, LAN toward Host-A, and its default route.
CPE	Fully pre-configured for basic forwarding. You will still configure the RDNSS/DNS advertisement on it in Activity 5.
Transit-Router	Fully pre-configured, including its side of the eBGP session and any policy needed to peer with AS-**U**00.
BB1 and BB2 IS-IS	IS-IS Level-2 is already running on BB1 and BB2. You'll bring up BB3 and BB4 in Activity 2 to complete the backbone.
BB2 iBGP	The core iBGP mesh sessions on BB2 (toward BB1, BB3, and BB4) are pre-loaded.
DC-Switch	All three VLANs (10, 21, 100) and their trunk/access ports are pre-configured.

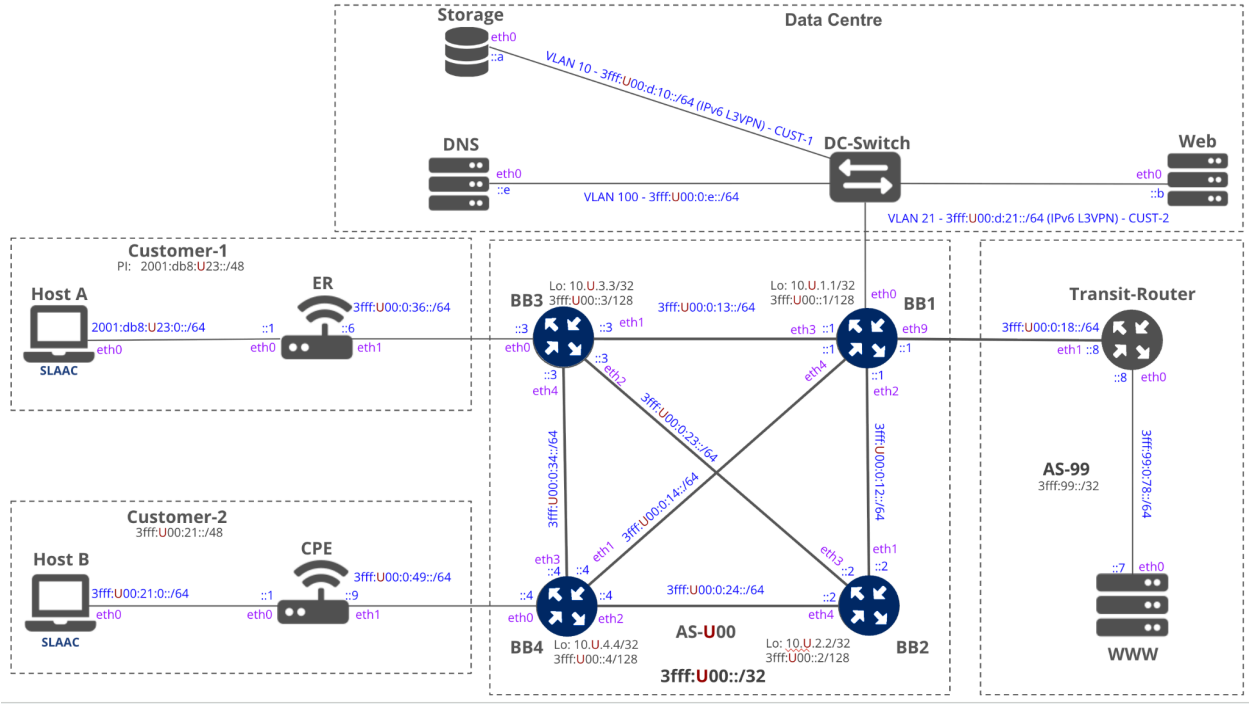
> **i Interface numbering convention**
>
> To make navigating the command line easier on every backbone router, interface ethN **always** faces router BBN (BB1's eth3 faces BB3, BB2's eth4 faces BB4, and so on). eth0 is always the customer- or Data Centre-facing port, and eth9 (on BB1 only) faces the Transit-Router.

What You Will Build

1. Activity 1: Register the AS-**U**00 addressing plan in phpIPAM.
2. Activity 2: Bring up IS-IS on BB3 and BB4 to complete the backbone (BB1 and BB2 are pre-staged).
3. Activity 3: Configure BGP: eBGP from BB1 to the Transit-Router, and iBGP on BB1, BB3, and BB4.
4. Activity 4: Layer SRv6 onto IS-IS, build the VRFs, and deliver an IPv6 L3VPN to each customer.
5. Activity 5: Configure DNS so both customers can resolve names for their Data Centre services.

Quick Reference Guide: Addressing Plan

Use this section throughout the booklet. You won't need to re-derive any address, just look it up here.



Core Inter-Router Links

Link	IPv6 Prefix (/64)	Side A	Side B
BB1 <-> BB3	3fff:U00:0:13::/64	BB1 eth3 (::1)	BB3 eth1 (::3)
BB1 <-> BB2	3fff:U00:0:12::/64	BB1 eth2 (::1)	BB2 eth1 (::2)
BB1 <-> BB4	3fff:U00:0:14::/64	BB1 eth4 (::1)	BB4 eth1 (::4)
BB3 <-> BB4	3fff:U00:0:34::/64	BB3 eth4 (::3)	BB4 eth3 (::4)
BB3 <-> BB2	3fff:U00:0:23::/64	BB3 eth2 (::3)	BB2 eth3 (::2)
BB4 <-> BB2	3fff:U00:0:24::/64	BB4 eth2 (::4)	BB2 eth4 (::2)

FRR Daemon and Linux Shells

In some of the lab exercises for this training course, you'll be moving between the FRR shell and the underlying Linux shell to type in or verify commands on the same router. One quick way to tell what shell you are in is to check the name in the shell prompt. The table below shows what the shell prompt looks like for User 1 on BB1 in either mode.

Shell	Prompt
Linux	u01-bb1:/#
FRR	u01-bb1#

The presence or absence of the colon and forward slash ":/" at the prompt is the most visible cue that lets you know what shell you are in on a router.

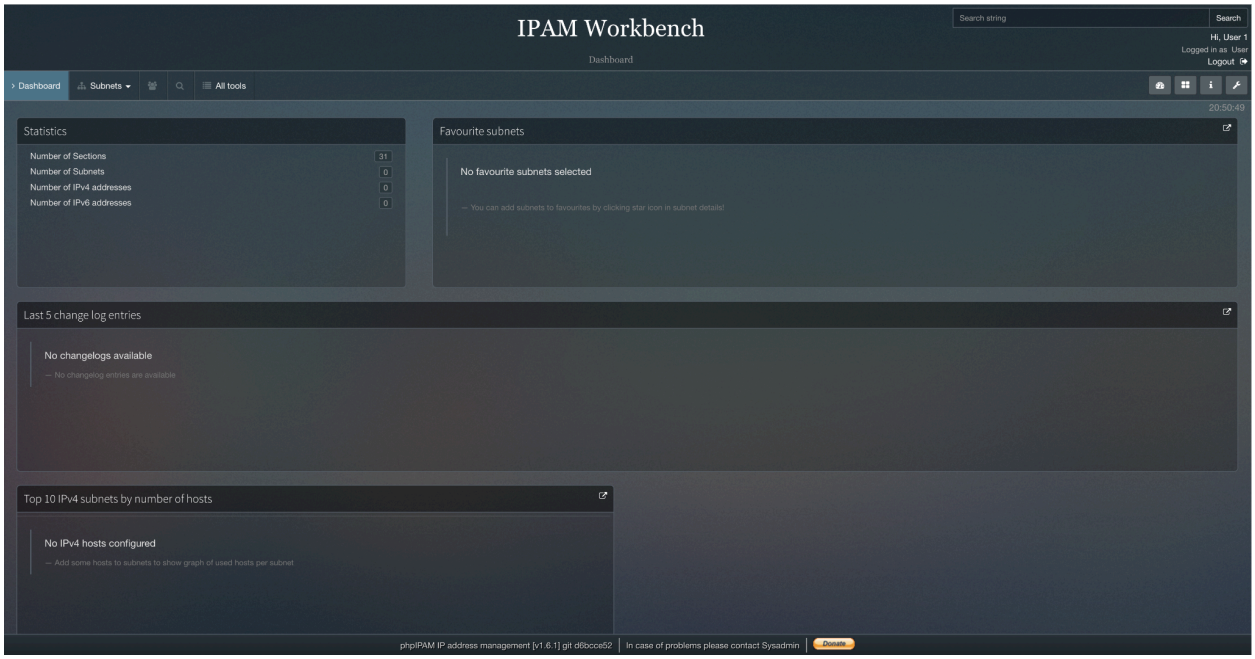
Activity 1: Build Your IPv6 Addressing Plan phpIPAM

Before you type a single routing command, it is good practice to record the addressing plan somewhere authoritative. In this activity you will register AS-U00's IPv6 allocation in phpIPAM, the same way you would before a real deployment:

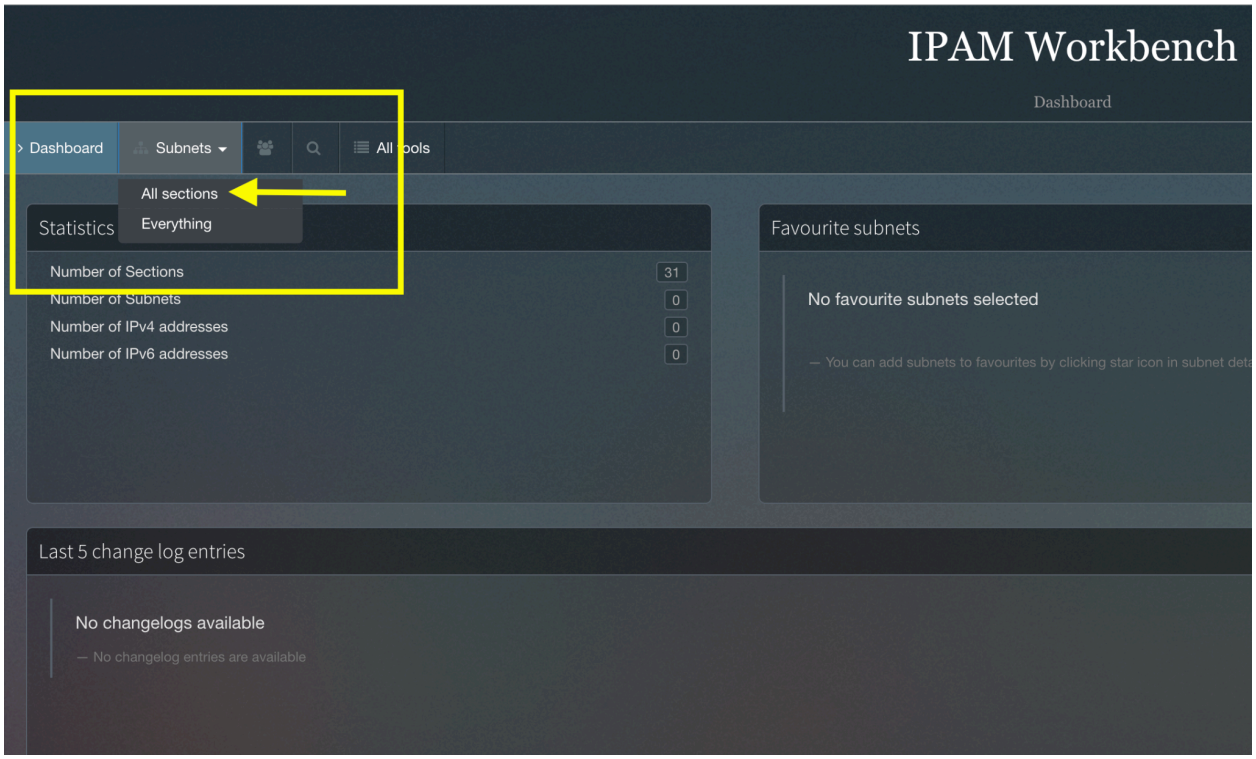
- create the parent allocation
- carve out nested subnets for each part of the network
- register a few key addresses
- check how much of your space is still free.

Activity 1.1: Register the AS-U00 Allocation

Sandbox Inc. has received the allocation **3fff:U00::/32 for AS-U00**, and you've been tasked with adding it to your IPAM tool, phpIPAM. Log in to phpIPAM with the credentials provided by your trainer. You should see the landing page below.



1. Click "**Subnets**", then "**All Sections**".



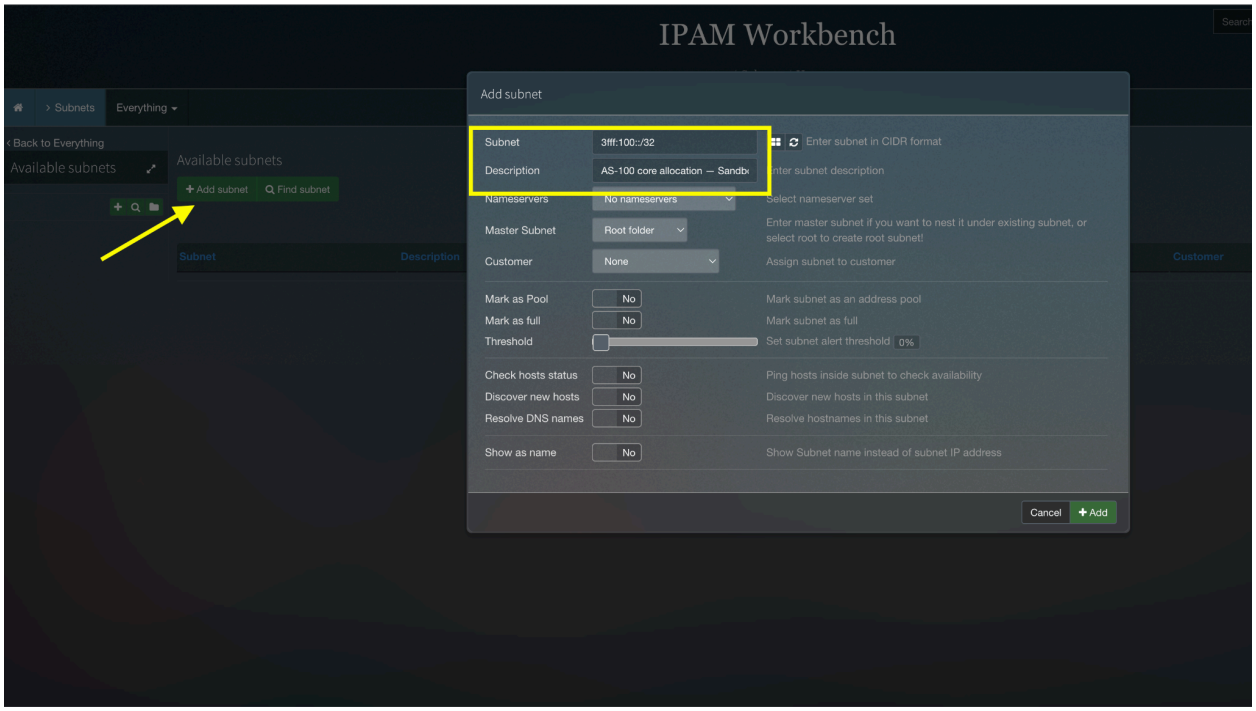
On this page you will see your username, shown as "**User X**". Click on your username to view your available resources.



The page you see will be similar to the one below. At this point you do not have any registered IP resources yet. Your first task is registering your allocation.



2. Click "**Add Subnet**" and enter your allocation in the form 3fff:**U**00::/32, substituting your own seat number for U. If your seat number is 3, your allocation is 3fff:300::/32. If your seat number is 24, your allocation is 3fff:2400::/32. The screenshots in this section show seat 1 working through the exercise, so they read 3fff:100::/32.



3. **Description:** "AS-**U**00 core allocation, Sandbox Inc."
4. Leave all other fields as default and save "**+Add**".

> ⚠ **Check the value before you save**
>
> This allocation is the parent of everything else you register in this activity, and every prefix in the rest of the booklet is derived from it. If you register the wrong block here, every nested subnet you create afterwards inherits the mistake and none of them will match the addresses configured on your pod. Confirm the number in your prefix is your seat number before saving.

Once your allocation is created successfully you will see a confirmation screen. You have now registered your allocation in your IPAM tool.

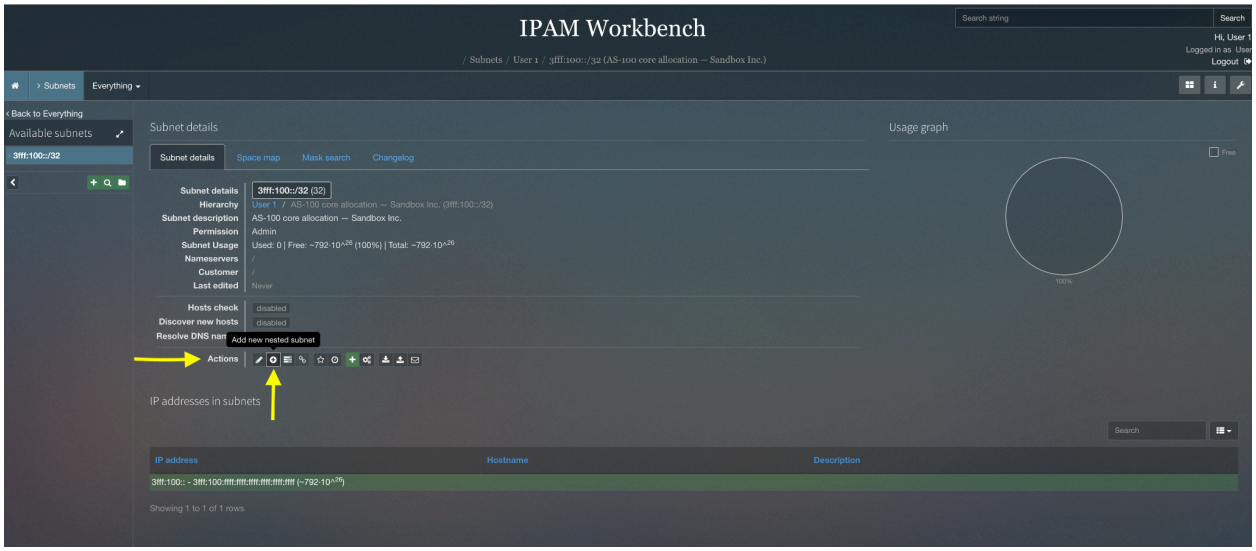
Activity 1.2: Create Nested Subnets for Each Part of the Network

Now, you need to create subnets under your allocation for the different parts of your network: the backbone, the Data Centre, the customer delegation, and the SRv6 locator range. In the real world, you will carve out all the blocks listed in the next table; mirroring the plan already in use on the live topology. However, due to time constraints, we will create the **backbone nested subnet only**. The same idea applies for all other nested subnets.

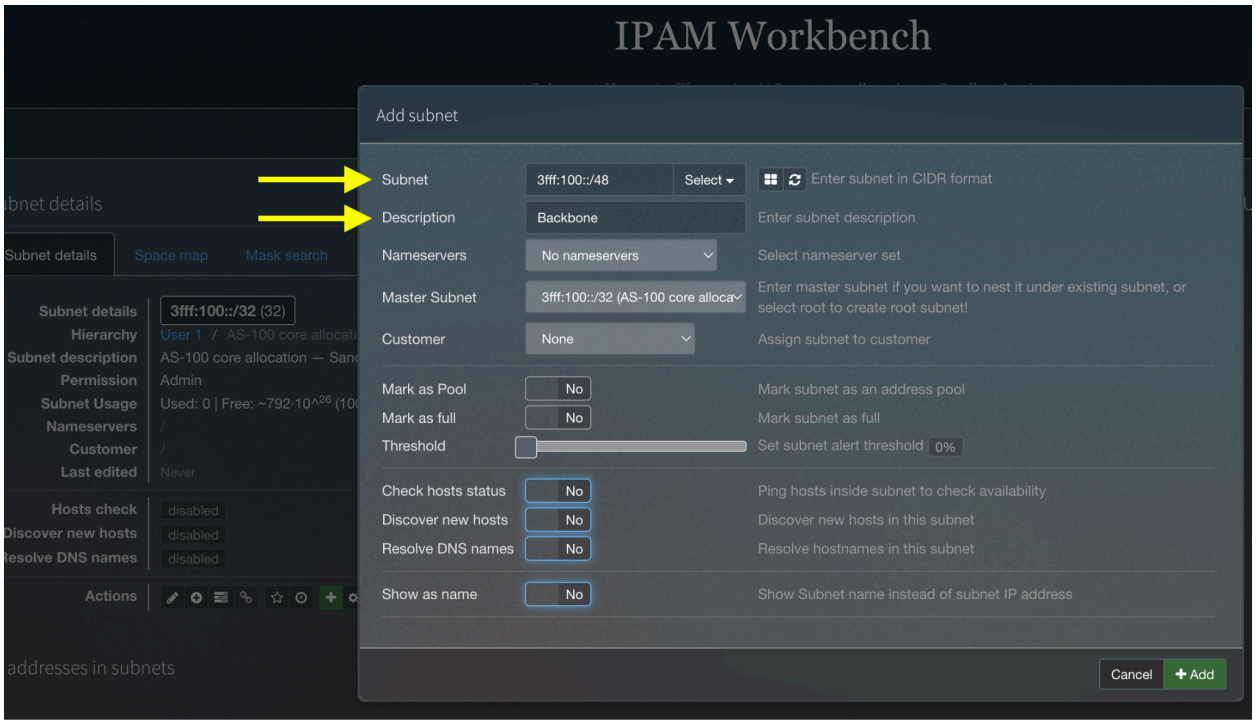
Nested Subnet	Purpose	Description to Enter
3fff:**U**00:0::/48	Backbone core links and loopbacks	Backbone network
3fff:**U**00:d::/48	Data Centre services, parent block for both customers' VPN prefixes	Data Centre network
3fff:**U**00:d:10::/64	Customer-1 Data Centre VPN service (VLAN 10), carved out of the Data Centre parent block above	Customer-1 Data Centre service
3fff:**U**00:d:21::/64	Customer-2 Data Centre VPN service (VLAN 21), carved out of the Data Centre parent block above	Customer-2 Data Centre service
3fff:**U**00:21::/48	Customer-2 delegated site space	Customer-2 network
3fff:**U**00:200::/48	SRv6 locator range, parent block for all four backbone router locators	SRv6 locator block
3fff:**U**00:200:1::/64	BB1 SRv6 locator, carved out of the SRv6 locator block above	BB1 SRv6 locator
3fff:**U**00:200:2::/64	BB2 SRv6 locator, carved out of the SRv6 locator block above	BB2 SRv6 locator
3fff:**U**00:200:3::/64	BB3 SRv6 locator, carved out of the SRv6 locator block above	BB3 SRv6 locator
3fff:**U**00:200:4::/64	BB4 SRv6 locator, carved out of the SRv6 locator block above	BB4 SRv6 locator

We will create the backbone prefix first and walk you through it in full.

1. Open the 3fff:U00::/32 parent prefix by clicking on it, then click the "**Add new nested subnet**" action. We will walk you through creating the first prefix.

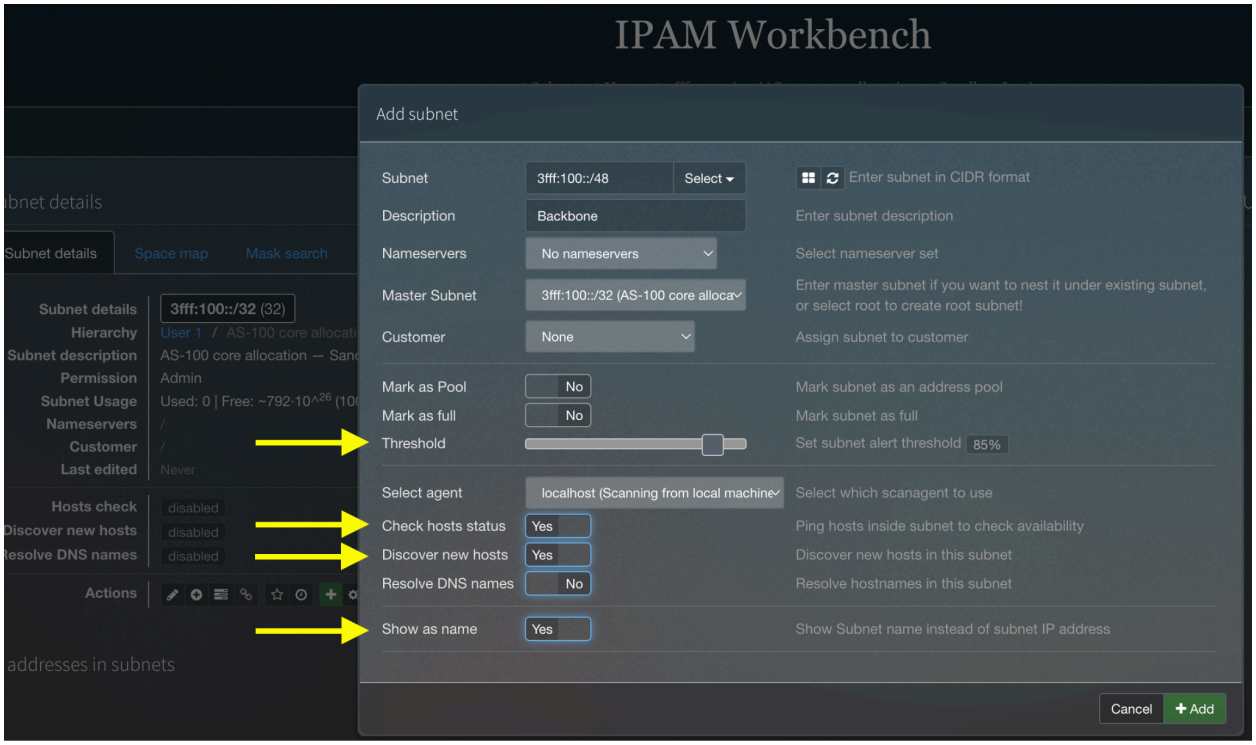


2. Enter the subnet and description as shown in the screenshot on the next page.

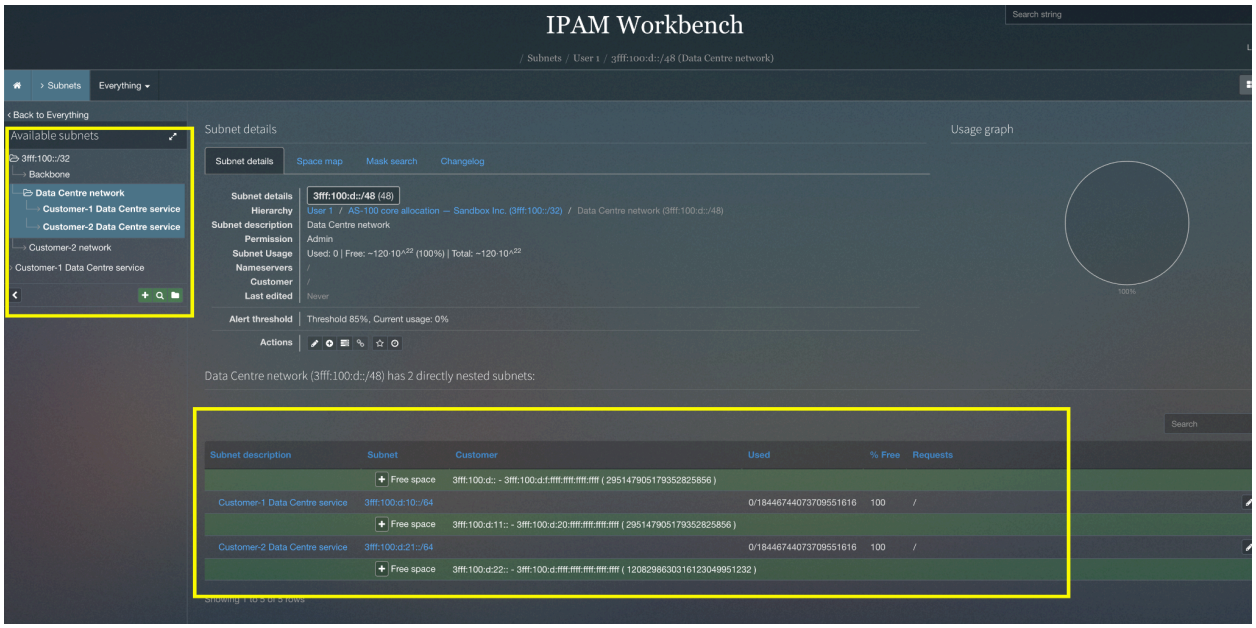


3. Optionally set a "**Threshold**" (for example 85% via the slider) to be notified when the subnet approaches full usage.
4. Toggle "**Check hosts status**" and "**Discover new hosts**" if you want phpIPAM to actively track host state on this subnet.

5. Toggle "**Show as name**" so the subnet displays with its description rather than the raw prefix, then save with the "**+Add**" button.



You can add the remaining nine nested subnets if you want the practice. The screenshot below shows what it looks like when you create some additional nested subnets in the IPAM.



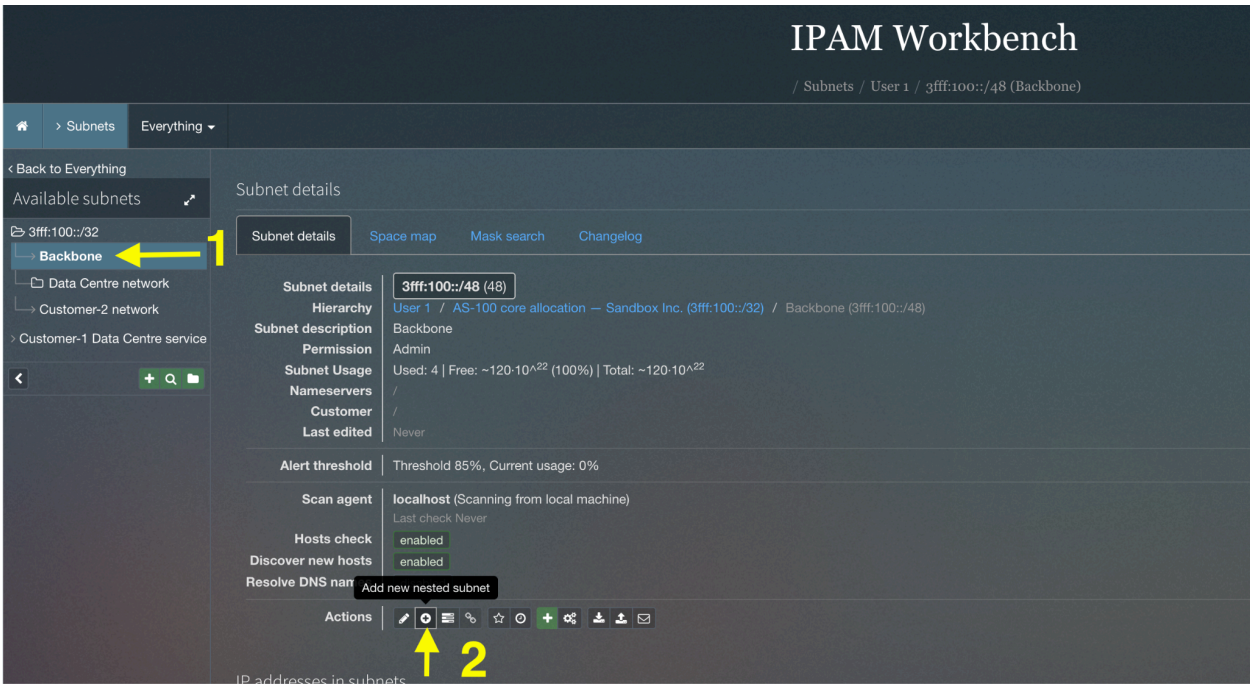
Notice that both customers' Data Centre VPN service prefixes go under the Data Centre network parent block, and all four router locators go under the SRv6 locator block, not directly under the top-level allocation.

> **i Note**
>
> Customer-1's own address space (2001:db8:**U**23::/48) is Provider Independent. It belongs to Customer-1's registry record, not AS-**U**00's allocation, so you will not register it here.

Activity 1.3: Register Key Addresses and Search for Them

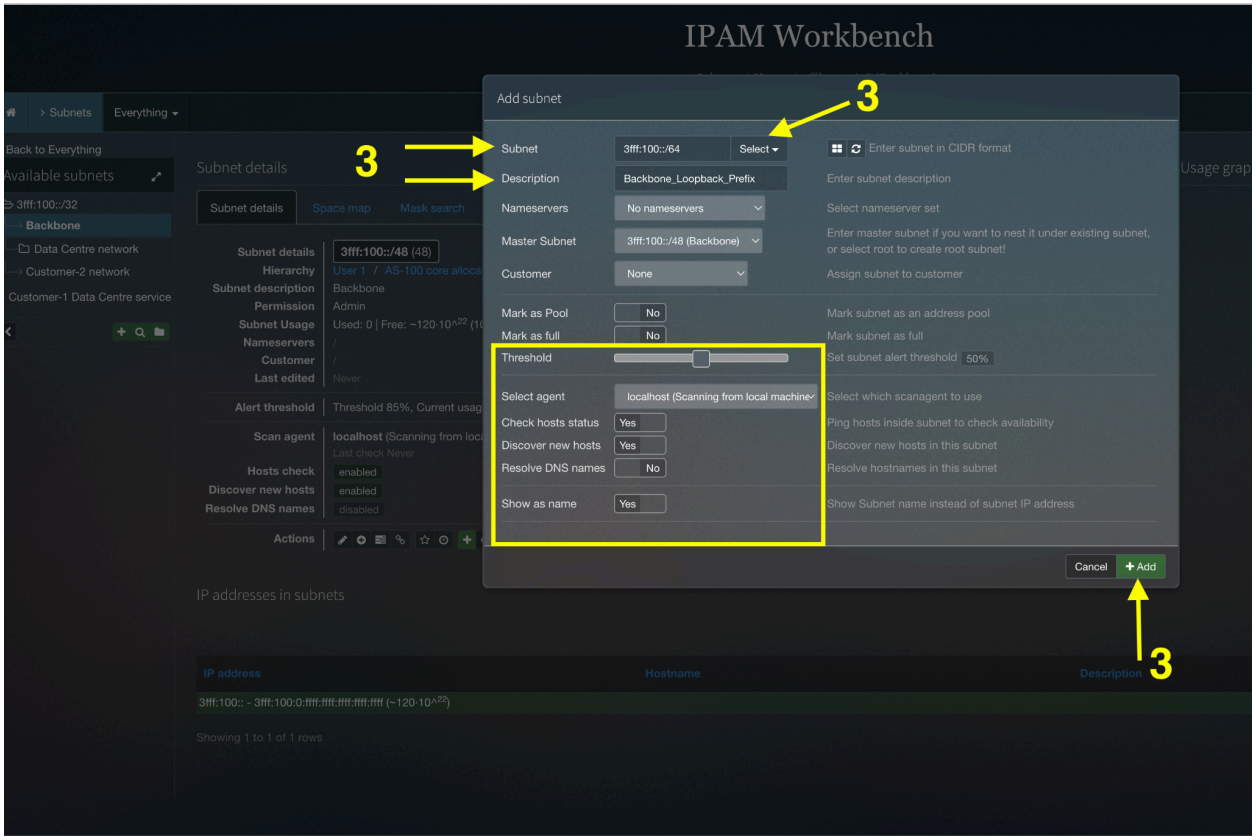
In this exercise, you will create an IP address record for one of your backbone router loopback interfaces and then search for it in the IPAM.

1. Open the "**Backbone network**" subnet (3fff:**U**00:0::/48) you created in the last activity.
2. Find the **"Actions"** row and click the **white** "**+**" icon to create a nested subnet.

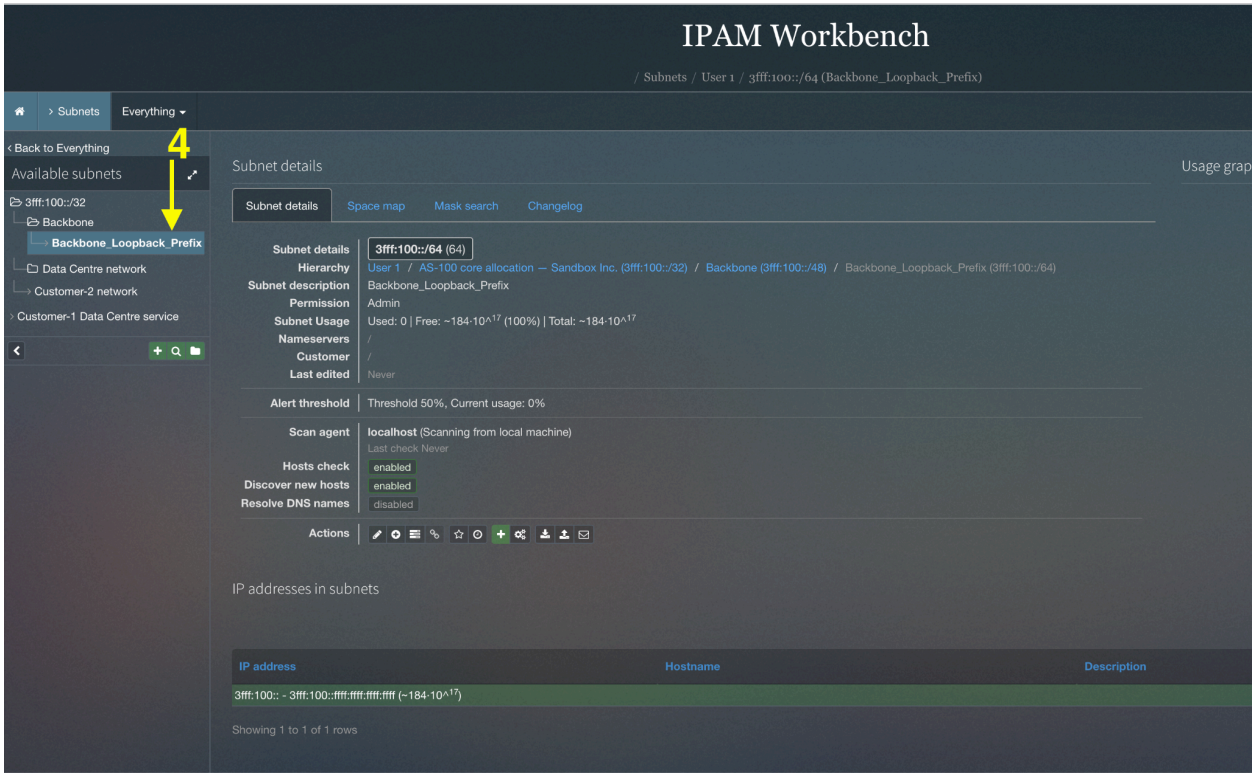


3. Click the "**Select**" drop-down in the "**Subnet**" field, scroll to the 3fff:**U**00::/64 prefix and select it. Set the Description to "Backbone_Loopback_Prefix", leave the remaining fields as shown in the first screenshot on page 18, then click "**+Add**".

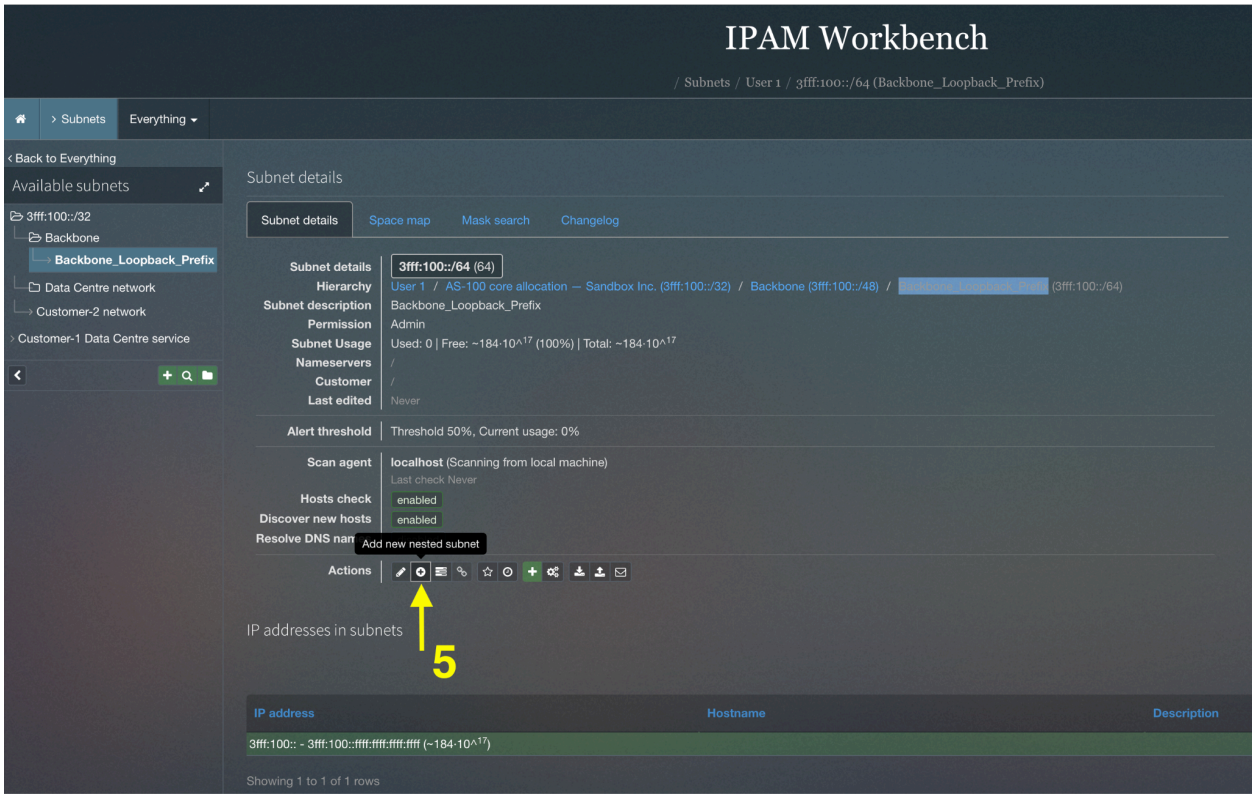
> **i A note on this step**
>
> Creating a dedicated nested subnet for your loopbacks reflects standard IPv6 address-planning practice. Registering loopbacks as bare hosts inside a larger shared subnet works too, but the nested-/128-subnet approach is the more conventionally documented one.



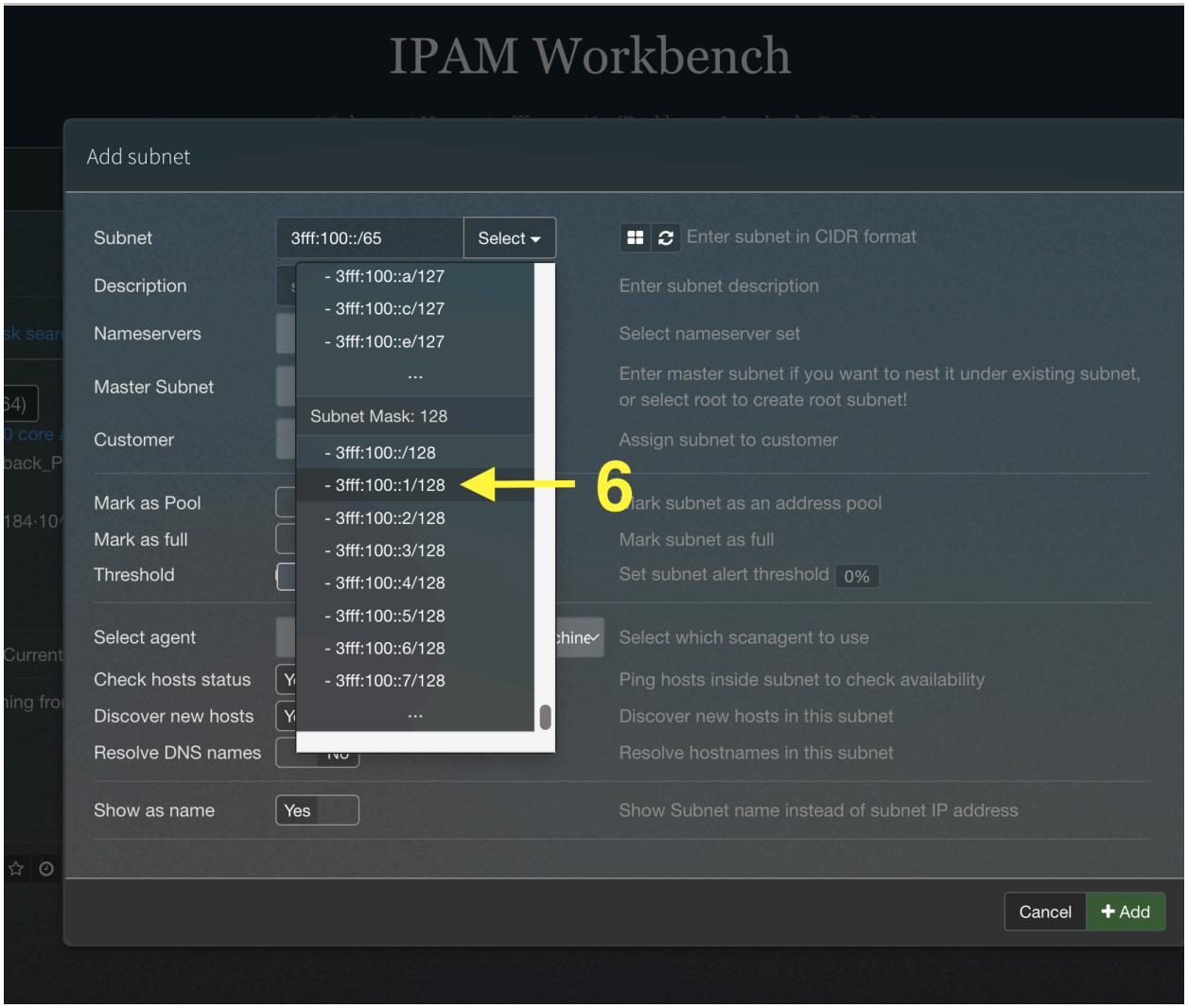
You should see a screen like the one below upon the successful creation of the nested subnet.

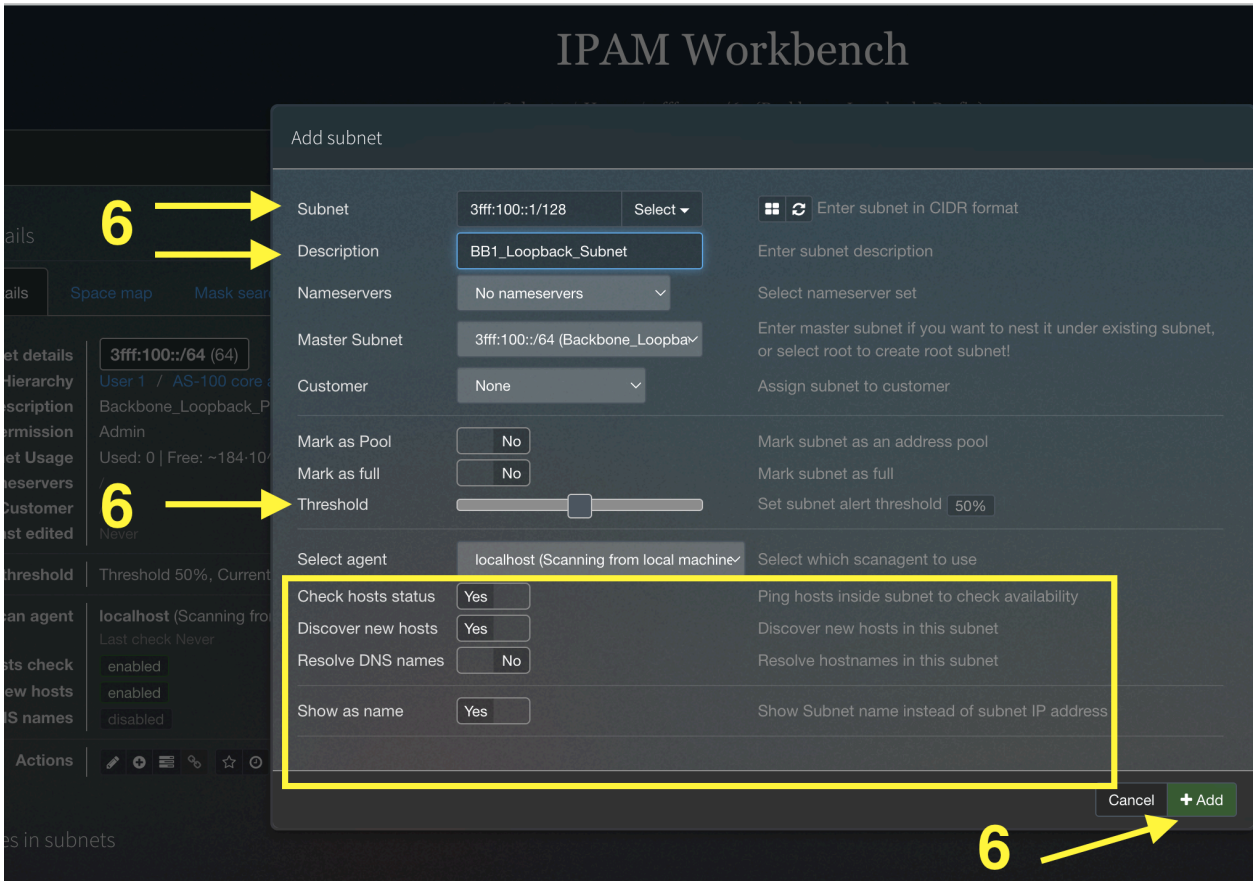


4. Click the newly created nested subnet (**Backbone_Loopback_Prefix**).
5. Find the "**Actions**" row and click the **white** "**+**" sign again. Hovering over it shows "**Add new nested subnet**".

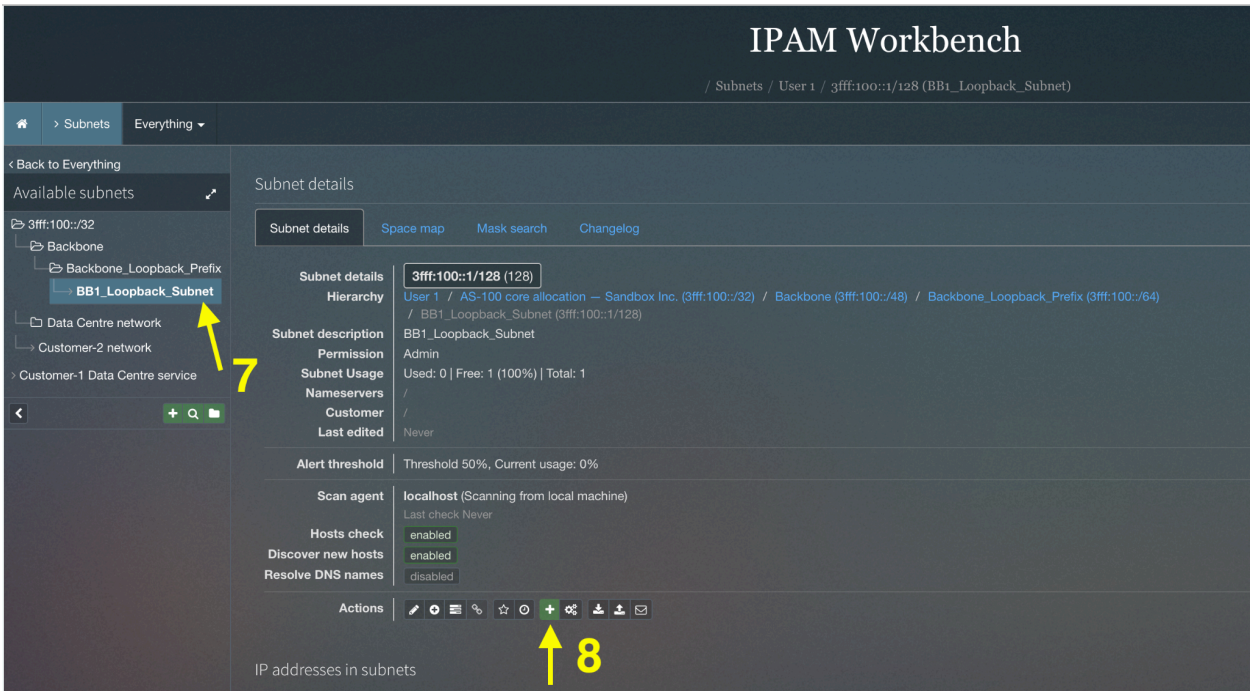


6. As before, click the "**Select**" drop-down in the "**Subnet**" field, scroll to the **3fff:U00::1/128** prefix and select it. Set Description to "BB1_Loopback_Subnet" and Threshold to 50%, leave the remaining fields at their values, then click "**+Add**".


IPAM Workbench
Add subnet
Subnet
3fff:100::/65
Select
Enter subnet in CIDR format
Description
Enter subnet description
Nameservers
Select nameserver set
Master Subnet
Enter master subnet if you want to nest it under existing subnet, or select root to create root subnet!
Customer
Assign subnet to customer
- 3fff:100::a/127
- 3fff:100::c/127
- 3fff:100::e/127
...
Subnet Mask: 128
- 3fff:100::/128
- 3fff:100::1/128
- 3fff:100::2/128
- 3fff:100::3/128
- 3fff:100::4/128
- 3fff:100::5/128
- 3fff:100::6/128
- 3fff:100::7/128
...
6
Mark as Pool
Mark subnet as an address pool
Mark as full
Mark subnet as full
Threshold
Set subnet alert threshold
0%
Select agent
Select which scanagent to use
Check hosts status
Ping hosts inside subnet to check availability
Discover new hosts
Discover new hosts in this subnet
Resolve DNS names
Resolve hostnames in this subnet
Show as name
Yes
Show Subnet name instead of subnet IP address
Cancel
+ Add



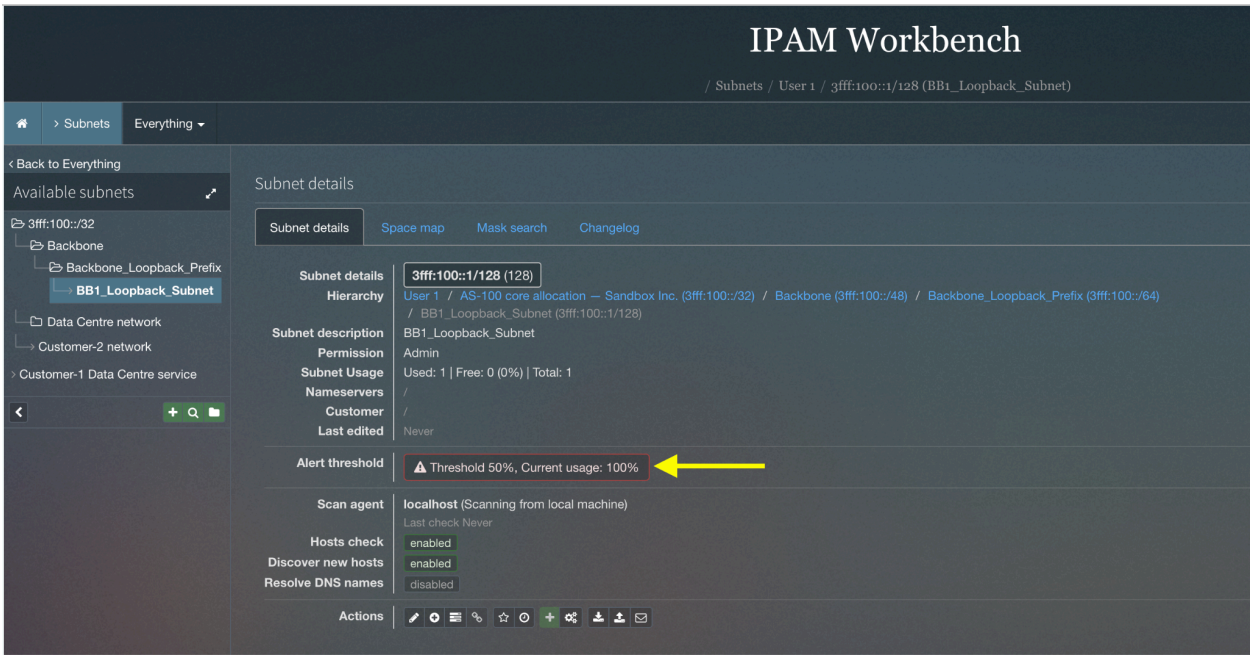
7. To create the actual IPv6 address for BB1's loopback interface, click the "**BB1_Loopback_Subnet**" you just created.



8. Add a new IP address by clicking the **green** "**+**" sign on the "Actions" row, with IP address 3fff:**U**00::1, hostname "BB1_lo", and description "BB1 loopback interface". Leave the other fields at their defaults and click "+Add IP".

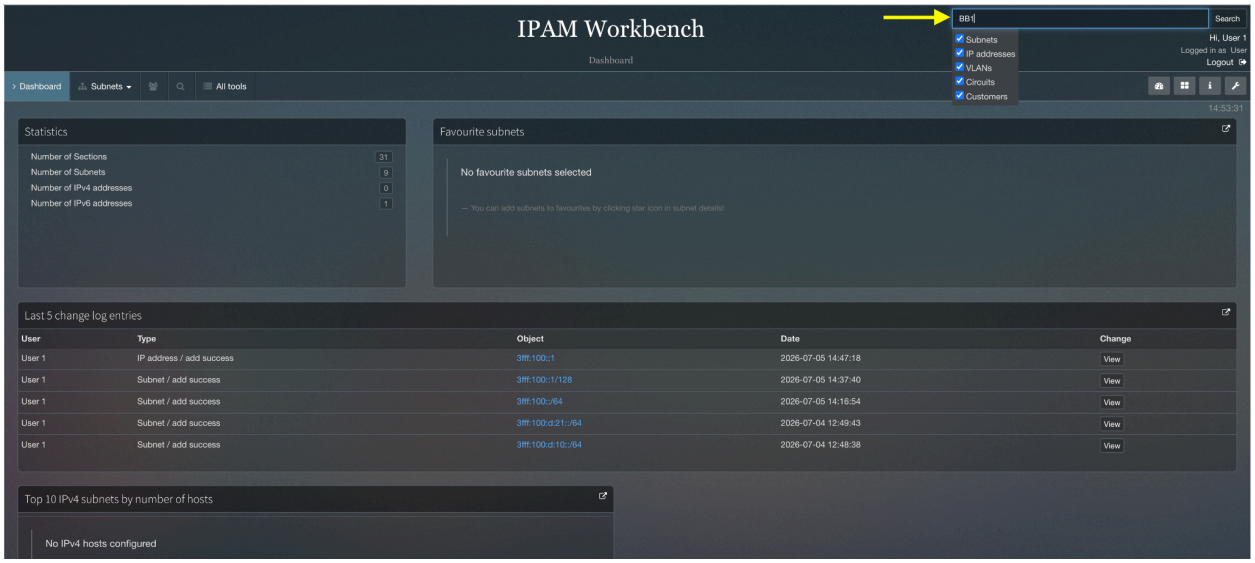
You should see a green "IP add successful" message. You will also see that the current usage for the subnet is 100%. Recall that a /128 subnet contains exactly one IPv6 address, which you have just created.



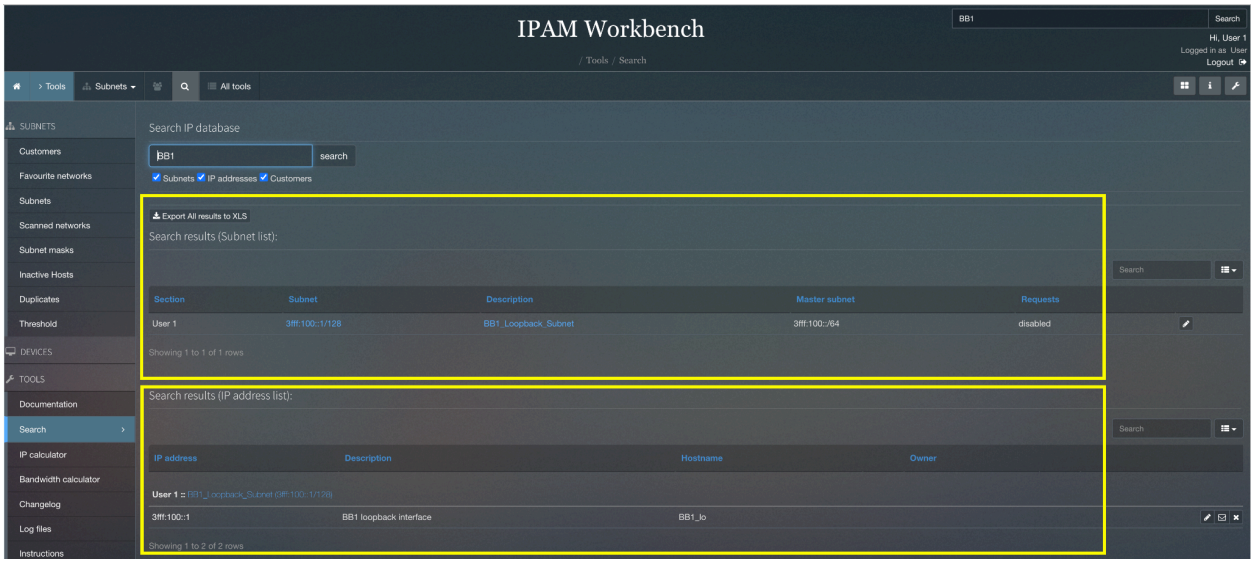


9. Repeat for BB2 (3fff:**U**00::2), BB3 (3fff:**U**00::3), and BB4 (3fff:**U**00::4), if you are not short on time.

For the final task in this activity, use the search bar in the top-right corner of the IPAM home page to search for "BB1" or "BB1_lo" and confirm the query returns the appropriate record.



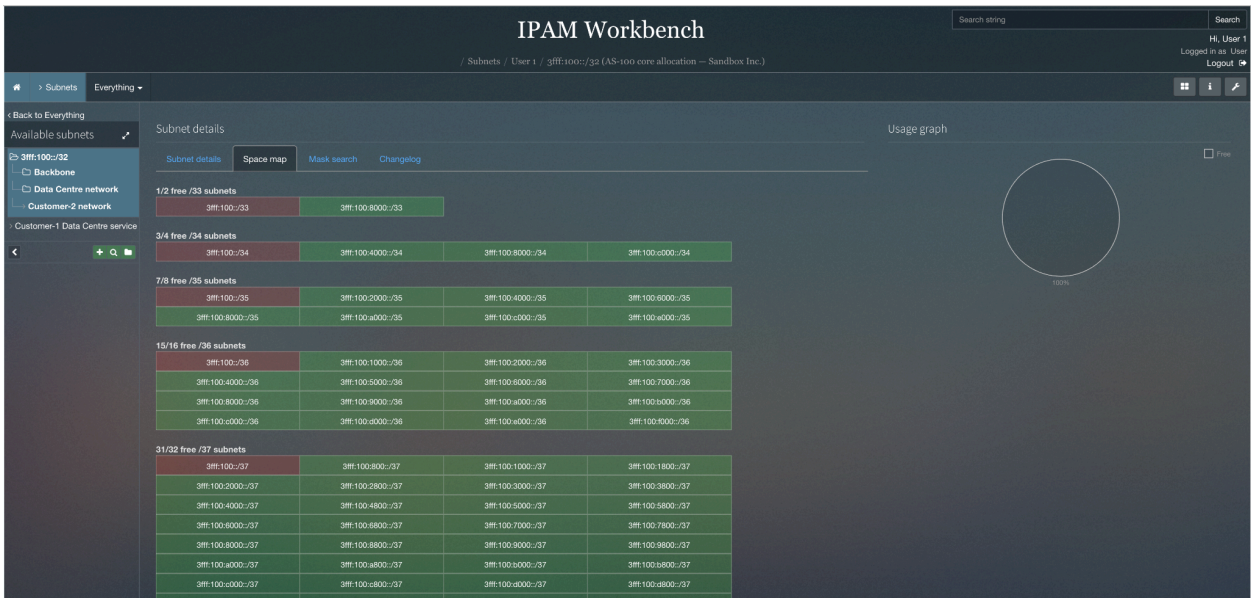
The search result should be as shown on the screen below.



Activity 1.4: Check Subnet Availability

phpIPAM also lets you check the availability of the subnets in your allocation.

1. Click "**Subnets**", then "**All Sections**".
2. Click your **username (User U)** to reach your available resources.
3. Click your parent allocation, 3fff:U00::/32.
4. In this view you can check "Subnet Usage" and see how much of your allocation is still free. Because a /32 is enormous relative to the handful of /48s you have carved from it, the figure will still be rounded to 100% free.
5. Click the "**Space Map**" tab to view your allocation's utilisation and availability at a glance.



Activity 1 Summary

- You registered AS-U00's top-level allocation and nested subnets for each functional area.
- You registered and searched for loopback addresses.
- You checked how much free address space you have remaining at a glance.

Activity 2: Configure IS-IS Inside the Backbone Network

None of the four backbone routers has any routing protocol running yet; well, almost none: **BB1** and **BB2** already have IS-IS Level-2 pre-staged for you, so you can focus this activity on **BB3** and **BB4**, the two routers you'll be returning to for BGP and the SRv6 L3VPN build in later activities. You'll bring up IS-IS Level-2 in a single area, running IPv6 reachability directly on the standard topology under a process named CORE. AS-**U**00 has no IPv4 anywhere in the underlay, so there's no need for the Multi-Topology extension that separates IPv4 and IPv6 into independent SPF runs.

Quick Reference Guide: Router Loopbacks and IS-IS NET

Router	IPv6 Loopback (/128)	IPv4 Loopback (/32)	IS-IS NET
BB1	3fff:**U**00::1	10.1.1.1	49.0001.0000.0000.0001.00
BB2	3fff:**U**00::2	10.1.2.2	49.0001.0000.0000.0002.00
BB3	3fff:**U**00::3	10.1.3.3	49.0001.0000.0000.0003.00
BB4	3fff:**U**00::4	10.1.4.4	49.0001.0000.0000.0004.00

Step 2.1: Check the Existing Configuration

Before configuring anything, confirm the starting state for yourself. Run on all four routers:

```
# show isis summary
```

BB1 and BB2 should already show an active IS-IS process with adjacencies; they are pre-staged. BB3 and BB4 should show an empty response. If BB3 or BB4 already show an IS-IS process, stop and flag it to your instructor before continuing.

The terminal output on BB1 or BB2 should be similar to the output below (executed on BB1).

BB1

```
u01-bb1# show isis summary
vrf             : default
Process Id      : 87
System Id       : 0000.0000.0001
Up time         : 02:16:55 ago
Number of areas : 1
Area CORE:
  Net: 49.0001.0000.0000.0001.00
  TX counters per PDU type:
```

```
    P2P IIH: 8652
     L2 LSP: 11
    L2 CSNP: 913
    L2 PSNP: 11
   LSP RXMT: 0
  RX counters per PDU type:
    P2P IIH: 2881
     L2 LSP: 11
    L2 CSNP: 913
    L2 PSNP: 12
  Drop counters per PDU type:
  Advertise high metrics: Disabled
  Level-2:
    LSP0 regenerated: 11
         LSPs purged: 0
    SPF:
      minimum interval  : 1
    IPv6 route computation:
      last run elapsed  : 00:06:28 ago
      last run duration : 192 usec
      run count         : 22
u01-bb1#
```

You should see the following output on **BB1** or **BB2:**

BB1

```
u01-bb1# show isis neighbor
Area CORE:
 System Id           Interface   L  State         Holdtime SNPA
 u01-bb2             eth2        2  Up            27       2020.2020.2020
u01-bb1#
```

On **BB3** or **BB4** you should get no output, as shown for BB3 below.

BB3

```
u01-bb3:/# vtysh

Hello, this is FRRouting (version 10.5.0_git).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

u01-bb3# show isis neighbor
u01-bb3#
```

Confirm **BB3** and **BB4** currently have no IPv6 reachability to the rest of the backbone. This ping should fail from either of them. Try pinging BB1's loopback address from BB3 or BB4.

```
# ping 3fff:U00::1          # from BB3 or BB4, before IS-IS exists there … in the FRR shell
```

```
u01-bb4# ping 3fff:U00::1
ping: connect: Network unreachable
u01-bb4#
```

Step 2.2: Configure IS-IS on BB3

BB3's core-facing interfaces are eth1 (to BB1), eth4 (to BB4), and eth2 (to BB2). eth0 (toward ER) is excluded: it is customer-facing, not a backbone link.

BB3

```
vtysh
configure terminal
!
router isis CORE
 is-type level-2-only
 net 49.0001.0000.0000.0003.00
 log-adjacency-changes
exit
!
interface lo
 ipv6 router isis CORE
 isis passive
exit
!
interface eth1
 ipv6 router isis CORE
 isis circuit-type level-2-only
 isis network point-to-point
exit
!
interface eth4
 ipv6 router isis CORE
 isis circuit-type level-2-only
 isis network point-to-point
exit
!
interface eth2
 ipv6 router isis CORE
 isis circuit-type level-2-only
 isis network point-to-point
exit
end
```

Command	Why You Are Typing It
is-type level-2-only	Restricts this process to Level-2 only; no Level-1 database, no L1/L2 boundary to manage. This backbone is a single flat area, so Level-1 would add nothing but overhead.
net 49.0001.0000.0000.0003.00	Sets the IS-IS area (49.0001) and this router's unique System ID (0000.0000.0003). Every router in the area must share the area number but have a distinct System ID; this is literally how IS-IS tells routers apart. **BB1** and **BB2's** pre-staged System IDs end in .0001 and .0002 respectively.
isis passive (on lo)	Advertises the loopback's /128 into the LSDB without attempting to establish an adjacency. A loopback has no neighbouring router to peer with, so passive is the only appropriate setting.

Command	Why You Are Typing It
isis circuit-type level-2-only (per interface)	Forces the interface to Level-2 only, matching the process-wide setting. This is a safety measure: if the global is-type is ever changed, this interface stays at Level-2 rather than dropping to Level-1.
isis network point-to-point	Stops the interface running a DR/BDR election. An election is only needed on networks that connect three or more routers. Every backbone link here connects exactly two routers, so the election is unnecessary and turning it off saves memory and CPU.
(no topology ipv6-unicast command)	AS-U00 is IPv6-only end to end. Multi-Topology IS-IS (RFC 5120) exists to compute separate paths when some links carry IPv4 and others do not. Here every link carries only IPv6, so the topologies would be identical and the extension is simply not needed. IS-IS computes IPv6 paths directly on the single standard topology.

i Multi-Topology extension

The topology ipv6-unicast command is never typed on any of the four routers, pre-staged or student-configured. This backbone is IPv6-only end to end, so there is no divergent IPv4/IPv6 topology for the Multi-Topology extension to solve, which is why it is absent everywhere rather than explicitly turned off on any one router.

Step 2.3: Configure IS-IS on BB4

BB4's core-facing interfaces are eth3 (to BB3), eth1 (to BB1), and eth2 (to BB2). eth0 (toward CPE) is excluded for the same reason.

BB4

```
vtysh
configure terminal
!
router isis CORE
 is-type level-2-only
 net 49.0001.0000.0000.0004.00
 log-adjacency-changes
exit
!
interface lo
 ipv6 router isis CORE
 isis passive
exit
!
interface eth3
 ipv6 router isis CORE
 isis circuit-type level-2-only
 isis network point-to-point
exit
```

```
!
interface eth1
 ipv6 router isis CORE
 isis circuit-type level-2-only
 isis network point-to-point
exit
!
interface eth2
 ipv6 router isis CORE
 isis circuit-type level-2-only
 isis network point-to-point
exit
end
```

Step 2.4: Verify IS-IS Adjacencies

On each of the four routers, confirm three Level-2 adjacencies in the Up state.

For example, on BB1:

```
# show isis neighbor
```

```
u01-bb1# show isis neighbor
Area CORE:
 System Id           Interface   L  State         Holdtime SNPA
 u01-bb2             eth2        2  Up            30       2020.2020.2020
 u01-bb3             eth3        2  Up            30       2020.2020.2020
 u01-bb4             eth4        2  Up            28       2020.2020.2020
u01-bb1#
```

This confirms the **control plane**: link-state information has been correctly exchanged and both sides agree.

Next, confirm full IPv6 reachability of routers' loopback addresses from multiple routers before moving on (in FRR shell prompt).

```
ping ipv6 3fff:U00::1
ping ipv6 3fff:U00::2
ping ipv6 3fff:U00::3
ping ipv6 3fff:U00::4
```

The ping tests confirm the **data plane** separately: the SPF-computed shortest path is actually installed and forwarding. It is possible, though unusual, for an adjacency to show up while a specific loopback is still unreachable, which is exactly why this booklet always asks you to check both rather than either alone.

Here is a sample ping output from **BB4** to **BB1's** loopback interface address.

BB4

```
u01-bb4# ping ipv6 3fff:100::1
PING 3fff:100::1 (3fff:100::1) 56 data bytes
64 bytes from 3fff:100::1: icmp_seq=1 ttl=64 time=0.043 ms
64 bytes from 3fff:100::1: icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from 3fff:100::1: icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from 3fff:100::1: icmp_seq=4 ttl=64 time=0.063 ms
^C
--- 3fff:100::1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3092ms
rtt min/avg/max/mdev = 0.043/0.058/0.072/0.010 ms
u01-bb4#
```

> **i If an adjacency won't come up**
>
> Check show isis neighbor on both ends of the link. Two routers showing the same System Id usually means a hostname wasn't set correctly, so confirm each router's hostname is unique before troubleshooting further.

Step 2.5: Inspect the IS-IS Link-State Database

The “show isis neighbor” command output tells you an adjacency formed. It does not tell you what the routers actually learned from each other. The link-state database (LSDB) is where that lives. Every router in the area holds an identical copy of it, which is what SPF runs against to compute the shortest path. Inspect the LSDB on BB1:

```
# show isis database
```

You should see one Link State PDU (LSP) per router in the area, four entries in total, one originated by each of BB1-BB4, including the two you didn't configure yourself. Pick BB3's own LSP and inspect it in detail using its LSP ID.

```
# show isis database detail uXX-bb3
```

You should see that each router's LSP lists its adjacencies and the IPv6 reachability it is originating. This is the same loopback and link information you registered by hand back in Activity 1, now being distributed automatically by the protocol. If a prefix is missing from every router's copy of the database, it will never be reachable no matter how many times you ping: the LSDB is the single source of truth every router computes its routing table from.

BB1

```
u01-bb1# show isis database
Area CORE:
IS-IS Level-2 link-state database:
LSP ID                 PduLen  SeqNumber   Chksum  Holdtime  ATT/P/OL
u01-bb1.00-00        *    165   0x0000000e  0xd1cb     386    0/0/0
u01-bb2.00-00             165   0x0000000e  0xdf95     421    0/0/0
u01-bb3.00-00             165   0x00000002  0x5216     412    0/0/0
u01-bb4.00-00             165   0x00000002  0xb0ad     443    0/0/0
    4 LSPs

u01-bb1#
```

BB1

```
u01-bb1# show isis database detail u01-bb3
Area CORE:
IS-IS Level-2 link-state database
LSP ID                 PduLen  SeqNumber   Chksum  Holdtime  ATT/P/OL
u01-bb3.00-00            165   0x00000002  0x5216     333    0/0/0
  Protocols Supported: IPv6
  Area Address: 49.0001
  Hostname: u01-bb3
  TE Router ID: 10.1.3.3
  Router Capability: 10.1.3.3 , D:0, S:0
  Extended Reachability: 0000.0000.0001.00 (Metric: 10)
  Extended Reachability: 0000.0000.0004.00 (Metric: 10)
  Extended Reachability: 0000.0000.0002.00 (Metric: 10)
  IPv4 Interface Address: 10.1.3.3
  IPv6 Reachability: 3fff:100::3/128 (Metric: 10)
  IPv6 Reachability: 3fff:100:0:13::/64 (Metric: 10)
  IPv6 Reachability: 3fff:100:0:34::/64 (Metric: 10)
  IPv6 Reachability: 3fff:100:0:23::/64 (Metric: 10)

u01-bb1#
```

i A note on what is not here yet

Notice that BB3's LSP advertises only its loopback and its three backbone links. There is no locator, and no SRv6 sub-TLV of any kind. That is expected at this stage: SRv6 rides inside IS-IS, but nothing has been configured to put it there. When you return to this same command in Activity 4, after configuring a locator, the LSP will carry an additional SRv6 Locator TLV. Comparing the two is a good way to see exactly what SRv6 adds to the IGP.

Finally, confirm the computed routes actually landed in the IPv6 Routing Information Base (RIB), marked with the IS-IS code letter (I).

BB3

```
# show ipv6 route isis
```

```
u01-bb3# show ipv6 route isis
Codes: K - kernel route, C - connected, L - local, S - static,
       R - RIPng, O - OSPFv3, I - IS-IS, B - BGP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric, t - Table-Direct,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

IPv6 unicast VRF default:
I>* 3fff:100::1/128 [115/20] via fe80::a8c1:abff:feb6:8c6b, eth1, weight 1, 00:17:41
I>* 3fff:100::2/128 [115/20] via fe80::a8c1:abff:feb1:c513, eth2, weight 1, 00:17:41
I>* 3fff:100::4/128 [115/20] via fe80::a8c1:abff:fe6b:46c4, eth4, weight 1, 00:17:21
I>* 3fff:100:0:12::/64 [115/20] via fe80::a8c1:abff:feb1:c513, eth2, weight 1, 00:17:41
  *                            via fe80::a8c1:abff:feb6:8c6b, eth1, weight 1, 00:17:41
I>* 3fff:100:0:14::/64 [115/20] via fe80::a8c1:abff:fe6b:46c4, eth4, weight 1, 00:17:21
  *                            via fe80::a8c1:abff:feb6:8c6b, eth1, weight 1, 00:17:21
I>* 3fff:100:0:24::/64 [115/20] via fe80::a8c1:abff:fe6b:46c4, eth4, weight 1, 00:17:21
  *                            via fe80::a8c1:abff:feb1:c513, eth2, weight 1, 00:17:21
u01-bb3#
```

Step 2.6: Watch IS-IS Hellos and CSNPs on the Wire

Everything so far has confirmed the result (adjacencies, database, routes). This step shows the actual protocol exchange that produced them. IS-IS is unusual among routing protocols in that it doesn't run over IP at all, it runs directly over the data-link layer (OSI CLNS), which is exactly what you should expect to see in the capture.

On **BB3**, capture on the link to **BB1** while the adjacency is up. Enter this command in the Linux shell on BB3.

```
u01-bb3# tcpdump -i eth1 -n -v isis
```

Press Ctrl+C to stop the command.

You will see two different PDU types on this link: IS-IS Hellos (IIH), which repeat every few seconds for as long as the adjacency stays up, and CSNPs (Complete Sequence Number PDUs), sent periodically alongside the Hellos. The output for a Link State PDU (LSP) sent by BB1 is shown below. To trigger it, use the "**clear isis neighbor**" vtysh command on BB1 while tcpdump is running on BB3.

```
BB1# clear isis neighbor
```

BB3

```
u01-bb3:/# tcpdump -i eth1 -n -v isis
-------------------------- Output omitted --------------------------
09:50:22.490935 IS-IS, length 165
        L2 LSP, hlen: 27, v: 1, pdu-v: 1, sys-id-len: 6 (0), max-area: 3 (0)
          lsp-id: 0000.0000.0001.00-00, seq: 0x00000010, lifetime:  1150s
          chksum: 0xcdcd (correct), PDU length: 165, Flags: [ L2 IS ]
            Protocols supported TLV #129, length: 1
              NLPID(s): IPv6 (0x8e)
            Area address(es) TLV #1, length: 4
              Area address (length: 3): 49.0001
            Hostname TLV #137, length: 7
              Hostname: u01-bb1
            IS-IS Router Capability TLV #242, length: 5
              Router-ID 10.1.1.1, Flags [none]
            Traffic Engineering Router ID TLV #134, length: 4
              Traffic Engineering Router ID: 10.1.1.1
            Extended IS Reachability TLV #22, length: 33
              IS Neighbor: 0000.0000.0002.00, Metric: 10, no sub-TLVs present
              IS Neighbor: 0000.0000.0003.00, Metric: 10, no sub-TLVs present5
              IS Neighbor: 0000.0000.0004.00, Metric: 10, no sub-TLVs present
            IPv4 Interface address(es) TLV #132, length: 4
              IPv4 interface address: 10.1.1.1
            IPv6 reachability TLV #236, length: 64
              IPv6 prefix: 3fff:100:0:12::/64, Distribution: up, Metric: 10, Internal
              IPv6 prefix: 3fff:100:0:13::/64, Distribution: up, Metric: 10, Internal
              IPv6 prefix: 3fff:100:0:14::/64, Distribution: up, Metric: 10, Internal
              IPv6 prefix: 3fff:100::1/128, Distribution: up, Metric: 10, Internal
-------------------------- Output omitted --------------------------
```

You should see BB1 describing itself to the rest of the network. Everything a router knows about its own position goes into one message like this, and every router floods one. The Hostname line names it. The three neighbour entries name BB2, BB3 and BB4, each with a cost of 10, so the network can work out the shortest path to anywhere. Below those are the addresses BB1 is advertising: its three backbone links and its loopback. Note that the router still carries an IPv4 address, 10.1.1.

Activity 2 Summary

- You confirmed the starting state before making any changes, including that BB1 and BB2 were already up.
- You brought up IS-IS Level-2 on BB3 and BB4, running IPv6 directly on the standard topology, completing the backbone.
- You verified full adjacency and loopback reachability across all four routers.
- You inspected the link-state database directly, rather than only trusting the computed result, and noted that it carries no SRv6 information yet.
- You captured IS-IS packets on the wire and saw that the protocol runs straight over Ethernet, with no IP header wrapping it.
- The backbone now has plain IPv6 reachability. No services yet; that comes in Activities 3 and 4.

Activity 3: Configure BGP in the Backbone Network

With IS-IS providing internal reachability, AS-**U**00 still has no route to anything outside itself. In this activity you'll configure eBGP from BB1 to the Transit-Router (AS-99), and a full iBGP mesh across BB1, BB3, and BB4. **iBGP on BB2 is already pre-loaded,** so once BB1, BB3, and BB4 join the mesh, all four routers will have a full iBGP mesh. BB1, BB3, and BB4 are also the three routers you'll come back to in Activity 4 to build the SRv6 VRFs. BB2 is a pure P router with no VRF role there, which is why it's the one pre-staged here rather than revisited later.

Quick Reference Guide: Transit / Internet Edge

Link	Prefix	Side A	Side B
BB1 <-> Transit-Router	3fff:**U**00:0:18::/64	BB1 eth9 (::1)	Transit-Router eth0 (::8)
Transit-Router <-> WWW	3fff:99:0:78::/64	Transit-Router eth1 (::8)	WWW eth0 (::7)

*eBGP: AS-**U**00 (BB1) <-> AS-99 (Transit-Router).*

Step 3.1: Check the Existing Configuration

Before touching BGP, confirm the starting state on each router rather than assuming it.

```
# show bgp ipv6 summary
```

On BB1, BB3, and BB4 you should see no BGP process at all (an explicit "BGP is not running" message, or an empty response). On BB2, by contrast, you should already see 3 iBGP neighbours, but sitting in Idle or Active rather than Established. That is expected: those sessions cannot complete until BB1, BB3, and BB4 have a BGP process to peer with, which you are about to build.

BB1

```
u01-bb1# show bgp ipv6 summary
% No BGP neighbors found in VRF default
u01-bb1#
```

BB2

```
u01-bb2# show bgp ipv6 summary

IPv6 Unicast Summary:
BGP router identifier 10.1.2.2, local AS number 100 VRF default vrf-id 0
BGP table version 0
```

```
RIB entries 0, using 0 bytes of memory
Peers 3, using 50 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
3fff:100::1     4        100         0       111        0    0    0    never       Active        0 N/A
3fff:100::3     4        100         0         0        0    0    0    never       Active        0 N/A
3fff:100::4     4        100         0         0        0    0    0    never       Active        0 N/A

Total number of neighbors 3
u01-bb2#
```

Step 3.2: Disable the Default IPv4 Unicast Address Family

By default FRR treats every BGP neighbour, even one configured with an IPv6 address, as a neighbor for IPv4 unicast routes. Turn that off on BB1, BB3, and BB4 before configuring any neighbour. Turning it off first keeps every neighbour you add afterward clean.

BB1, BB3, and BB4

```
vtysh
configure terminal
router bgp U00
 no bgp default ipv4-unicast
end
```

Step 3.3: Configure eBGP on BB1 to the Transit-Router

The Transit-Router side of this session is already configured, so you only need to configure BB1.

BB1

```
vtysh
configure terminal
router bgp U00
 neighbor 3fff:U00:0:18::8 remote-as 99
 address-family ipv6 unicast
  neighbor 3fff:U00:0:18::8 activate
 exit-address-family
end
```

Command	Why You Are Typing It
neighbor 3fff:U00:0:18::8 remote-as 99	Declares the Transit-Router as an eBGP peer. remote-as 99 differs from AS-U00's own number, which is what makes this an eBGP rather than iBGP session and triggers eBGP's default behaviours, such as AS-path prepending on advertisement.
neighbor … activate (under address-family ipv6 unicast)	A neighbour statement alone only opens a TCP session. No routes flow in any address family until you explicitly

Command	Why You Are Typing It
	activate it, which is what actually turns the session into something useful.

Check the session. At this point it should be up but exchanging nothing, because no policy has been applied yet.

BB1

```
# show bgp ipv6 summary
```

```
u01-bb1# show bgp ipv6 summary

IPv6 Unicast Summary:
BGP router identifier 10.1.1.1, local AS number 100 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 17 KiB of memory

Neighbor         V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
3fff:100:0:18::8 4         99         4         3        0    0    0 00:00:20     (Policy) (Policy) N/A

Total number of neighbors 1
u01-bb1#
```

Step 3.4: Accept Routes from the Transit Network (IN policy)

BB1

```
configure terminal
ipv6 prefix-list ANY_v6 seq 10 permit 2000::/3 le 48
route-map ANY_IN_v6 permit 100
 match ipv6 address prefix-list ANY_v6
exit
router bgp U00
 address-family ipv6 unicast
  neighbor 3fff:U00:0:18::8 route-map ANY_IN_v6 in
 exit-address-family
end
```

i Why /48

/48 is the smallest prefix length that is routed globally in the IPv6 Internet as a best practice, so this filter simply matches best-practice operator behaviour.

Step 3.5: Advertise AS-U00's Own Space (OUT policy)

You now need to advertise AS-U00's backbone block and Customer-1's PI space outward. Because BGP only advertises exact matches, you first create Null0 static routes for the aggregates you intend to announce.

BB1

```
configure terminal
ipv6 route 3fff:U00::/32 Null0
ipv6 route 2001:db8:U23::/48 Null0
!
ipv6 prefix-list ASU00_NETW_v6 seq 10 permit 3fff:U00::/32
ipv6 prefix-list ASU00_NETW_v6 seq 20 permit 2001:db8:U23::/48
!
route-map ASU00_NETW_OUT_v6 permit 100
 match ipv6 address prefix-list ASU00_NETW_v6
exit
!
router bgp U00
 address-family ipv6 unicast
  redistribute static
  neighbor 3fff:U00:0:18::8 route-map ASU00_NETW_OUT_v6 out
 exit-address-family
end
```

Command	Why You Are Typing It
ipv6 route … Null0	BGP only advertises prefixes that exist in the routing table as an exact match. Without a route to the aggregate itself, there is nothing for redistributing static to pick up. Null0 is a discard route: it exists purely to give BGP something to advertise, not to forward traffic, since real traffic is always matched by a longer, more specific route first.
ipv6 prefix-list AS100_NETW_v6	Defines exactly which prefixes are allowed to be advertised outward. Without this, an accidental redistribute static could leak routes you never intended to announce.
route-map AS100_NETW_OUT_v6	Wraps the prefix-list into something a neighbour statement can apply. Prefix-lists filter addresses; route-maps attach that filtering, and any other policy, to a specific direction on a specific neighbour.
redistribute static	Pulls the Null0 routes into BGP's candidate table in the first place. This is the step that makes them exist inside BGP at all, before the route-map even gets a chance to filter them outward.
neighbor … route-map … out	Applies the filter specifically to the OUT direction toward the Transit-Router. IN and OUT are independent policies in BGP, so this has zero effect on what you configured in Step 3.4.

i About the customer prefixes

Customer-1's PI space (2001:db8:**U**23::/48) needs its own announcement: it is a completely separate top-level block from AS-**U**00's own allocation, so the aggregate says nothing about it. Customer-2's delegated space is different. It does not get its own line here, because 3fff:**U**00:21::/48 is carved directly out of 3fff:**U**00::/32 and the aggregate you are already announcing covers it completely. Announcing a more-specific prefix externally only makes sense when there is a concrete reason such as multihoming, traffic engineering, or an independent RPKI ROA, none of which apply here.

Check the eBGP session on BB1 again. At this point it should be up and exchanging routes, because you have just applied a policy.

```
u01-bb1# show bgp ipv6 summary

IPv6 Unicast Summary:
BGP router identifier 10.1.1.1, local AS number 100 VRF default vrf-id 0
BGP table version 3
RIB entries 3, using 384 bytes of memory
Peers 1, using 17 KiB of memory

Neighbor         V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
3fff:100:0:18::8 4         99        13        11        3    0    0 00:06:52            1        2 N/A

Total number of neighbors 1
u01-bb1#
```

Step 3.6: Configure iBGP on BB1, BB3, and BB4

BB2 already has its iBGP sessions to BB1, BB3, and BB4 pre-loaded. You need to configure BB1, BB3, and BB4 so the mesh is complete: each of these three needs a neighbor statement toward BB2 and toward the other two.

BB1

```
configure terminal
router bgp U00
 neighbor 3fff:U00::2 remote-as U00
 neighbor 3fff:U00::2 update-source lo
 neighbor 3fff:U00::3 remote-as U00
 neighbor 3fff:U00::3 update-source lo
 neighbor 3fff:U00::4 remote-as U00
 neighbor 3fff:U00::4 update-source lo
 address-family ipv6 unicast
  neighbor 3fff:U00::2 activate
  neighbor 3fff:U00::2 next-hop-self
  neighbor 3fff:U00::3 activate
  neighbor 3fff:U00::3 next-hop-self
  neighbor 3fff:U00::4 activate
  neighbor 3fff:U00::4 next-hop-self
 exit-address-family
end
```

BB3

```
configure terminal
router bgp U00
 bgp router-id 10.U.3.3
 neighbor 3fff:U00::1 remote-as U00
 neighbor 3fff:U00::1 update-source lo
 neighbor 3fff:U00::2 remote-as U00
 neighbor 3fff:U00::2 update-source lo
 neighbor 3fff:U00::4 remote-as U00
 neighbor 3fff:U00::4 update-source lo
 address-family ipv6 unicast
  neighbor 3fff:U00::1 activate
  neighbor 3fff:U00::1 next-hop-self
  neighbor 3fff:U00::2 activate
  neighbor 3fff:U00::2 next-hop-self
  neighbor 3fff:U00::4 activate
  neighbor 3fff:U00::4 next-hop-self
 exit-address-family
end
```

BB4

```
configure terminal
router bgp U00
 bgp router-id 10.U.4.4
 neighbor 3fff:U00::1 remote-as U00
 neighbor 3fff:U00::1 update-source lo
 neighbor 3fff:U00::2 remote-as U00
 neighbor 3fff:U00::2 update-source lo
 neighbor 3fff:U00::3 remote-as U00
 neighbor 3fff:U00::3 update-source lo
 address-family ipv6 unicast
  neighbor 3fff:U00::1 activate
  neighbor 3fff:U00::1 next-hop-self
  neighbor 3fff:U00::2 activate
  neighbor 3fff:U00::2 next-hop-self
  neighbor 3fff:U00::3 activate
  neighbor 3fff:U00::3 next-hop-self
 exit-address-family
end
```

Command	Why You Are Typing It
bgp router-id	Every BGP speaker needs a unique 32-bit router ID, which it carries in its OPEN message to identify itself to peers. FRR normally picks this automatically from the highest IPv4 address on the router. **On an IPv6-only router there is no IPv4 address to borrow**, so the ID defaults to 0.0.0.0 and no BGP session will ever come up. Setting it manually fixes this. The value only has to be unique within the network; it does not need to be a real, reachable address
update-source lo	iBGP sessions conventionally source from the loopback rather than a physical interface. Loopback interfaces never go down as long as the router is up, so the session survives any single link flapping, as long as IS-IS still has an alternate path to the other loopback.

Command	Why You Are Typing It
next-hop-self	Without this, BB1 would advertise routes learned from the Transit-Router with the Transit-Router's own next-hop address, which BB3 and BB4 have no direct route to. next-hop-self rewrites the next hop to BB1's own loopback before advertising it onward, so every iBGP peer only ever needs a route to another AS-U00 loopback, something IS-IS already guarantees.

Step 3.7: Verify Session State and the BGP Table

On BB1:

```
# show bgp ipv6 summary
```

You should see 4 neighbours (3 iBGP plus 1 eBGP). On **BB2**, **BB3**, and **BB4** you should see 3 iBGP neighbors each, all now established. You should not see "Active" for any BGP speaker under the State/PfxRcd column.

BB1

```
u01-bb1# show bgp ipv6 summary

IPv6 Unicast Summary:
BGP router identifier 10.1.1.1, local AS number 100 VRF default vrf-id 0
BGP table version 3
RIB entries 3, using 384 bytes of memory
Peers 4, using 66 KiB of memory

Neighbor         V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
3fff:100::2      4        100         7         9        3    0    0 00:04:02            0        3 FRRouting/10.5.0_git
3fff:100::3      4        100         5        10        3    0    0 00:02:14            0        3 FRRouting/10.5.0_git
3fff:100::4      4        100         3         8        3    0    0 00:00:45            0        3 FRRouting/10.5.0_git
3fff:100:0:18::8 4         99        23        21        3    0    0 00:16:04            1        2 N/A

Total number of neighbors 4
u01-bb1#
```

The "**show bgp ipv6 summary**" command only tells you sessions are up. To inspect the actual table BGP built, use show bgp ipv6. Execute this on **BB1** and **BB3**

BB1

```
u01-bb1# show bgp ipv6
BGP table version is 3, local router ID is 10.1.1.1, vrf id 0
Default local pref 100, local AS 100
Status codes:  s suppressed, d damped, h history, u unsorted, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
```

```
 *>  2000::/3         fe80::a8c1:abff:fe00:a2ea
                                           0             0 99 ?
 *>  2001:db8:123::/48
                     ::                    0         32768 ?
 *>  3fff:100::/32    ::                     0         32768 ?

Displayed 3 routes and 3 total paths
u01-bb1#
```

BB3

```
u01-bb3# show bgp ipv6
BGP table version is 3, local router ID is 10.1.3.3, vrf id 0
Default local pref 100, local AS 100
Status codes:  s suppressed, d damped, h history, u unsorted, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 *>i 2000::/3         3fff:100::1              0    100      0 99 ?
 *>i 2001:db8:123::/48
                     3fff:100::1              0    100      0 ?
 *>i 3fff:100::/32    3fff:100::1              0    100      0 ?

Displayed 3 routes and 3 total paths
u01-bb3#
```

Then look at one specific eBGP-learned route in detail, including its AS-path and where it came from.

```
BB1# show bgp ipv6 2000::/3
```

The eBGP-learned aggregate route should show AS-path 99 and a next hop of the Transit-Router's address. Routes you are advertising outward (AS-**U**00's own block and Customer-1's PI space from Step 3.5) should show an empty AS-path: they originate in AS-**U**00, so there is nothing to prepend yet. If you see AS-path 99 on a route you expected to be locally originated, you have likely got IN and OUT policy crossed on the wrong neighbour.

BB1

```
--------------------------- External Route ---------------------------
u01-bb1# show bgp ipv6 2000::/3
BGP routing table entry for 2000::/3, version 1
Paths: (1 available, best #1, table default)
  Advertised to peers:
  3fff:100::2 3fff:100::3 3fff:100::4
  99
    3fff:100:0:18::8 from 3fff:100:0:18::8 (10.0.8.8)
```

```
    (fe80::a8c1:abff:fe00:a2ea) (used)
      Origin incomplete, metric 0, valid, external, best (First path received)
      Last update: Wed Sep  2 10:54:28 2026
u01-bb1#
```

BB1

```
--------------------------- Local Route --------------------------
u01-bb1# show bgp ipv6 3fff:100::/32
BGP routing table entry for 3fff:100::/32, version 3
Paths: (1 available, best #1, table default)
  Advertised to peers:
  3fff:100::2 3fff:100::3 3fff:100::4 3fff:100:0:18::8
  Local
    :: from :: (10.1.1.1)
      Origin incomplete, metric 0, weight 32768, valid, sourced, best (First path received)
      Last update: Wed Sep  2 10:56:31 2026
u01-bb1#
```

Confirm the Transit-Router is now sending and receiving prefixes (non-zero PfxRcd and PfxSnt in the show bgp ipv6 summary output) on BB1, and that you can ping the WWW server from BB3 or BB4.

BB4

```
# ping 3fff:99:0:78::7
```

```
u01-bb4# ping 3fff:99:0:78::7
PING 3fff:99:0:78::7 (3fff:99:0:78::7) 56 data bytes
64 bytes from 3fff:99:0:78::7: icmp_seq=1 ttl=62 time=0.182 ms
64 bytes from 3fff:99:0:78::7: icmp_seq=2 ttl=62 time=0.097 ms
64 bytes from 3fff:99:0:78::7: icmp_seq=3 ttl=62 time=0.091 ms
^C
--- 3fff:99:0:78::7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.091/0.123/0.182/0.041 ms
u01-bb4#
```

You have just proved that a successful ping from BB3 or BB4 all the way to the WWW server crosses three separate hops of trust: IS-IS got the packet from BB3 or BB4 to BB1 (Activity 2), iBGP carried the Transit-Router's routes from BB1 to BB3 and BB4 with next-hop-self keeping the next hop resolvable via IS-IS the whole way, and eBGP is what got those routes into AS-**U**00 in the first place. If any one of those three pieces were broken, this ping would fail even though the other two were fine.

> ⚠ **If this ping fails, read the failure before changing anything.**

Activity 3 Summary

- You confirmed the starting state: no BGP on BB1, BB3 and BB4, iBGP pre-staged but Active on BB2.
- You configured eBGP between BB1 and the Transit-Router, with a policy to accept and advertise routes.
- You completed the iBGP full mesh across AS-**U**00 by configuring BB1, BB3, and BB4.
- You inspected the actual BGP table, not just session state, to confirm what was learned and advertised.
- You verified reachability to the simulated Internet (WWW, AS-99).

Activity 4: Configure IPv6 L3VPN over SRv6

This is where the backbone starts doing what Sandbox Inc. is actually selling: private connectivity. In this activity, you will:
Part A: Enable SRv6 in IS-IS
Part B: Connect Customer-1's private service in the Data Centre (Storage)
Part C: Connect Customer-2's private service in the Data Centre (WEB)
Part D: Give the shared DNS server its own small VRF
Part E: Verify BGP VPNv6 Sessions and SID Allocation

Quick Reference Guide: SRv6 Locators

This lab uses uncompressed SRv6 SIDs. These are full 128-bit SIDs as defined structurally in RFC 8986, with no compression flavour applied. FRR selects the field layout with format uncompressed-f4024, a widely used convention that sets a **40-bit locator block**, a **24-bit node ID** and a **16-bit function**, making each router's locator a /64.

Router	SRv6 Locator (/64)	Node ID	Role
BB1	3fff:**U**00:200:1::/64	1	Egress PE (Data Centre)
BB2	3fff:**U**00:200:2::/64	2	P router (Transit)
BB3	3fff:**U**00:200:3::/64	3	Ingress PE (Customer-1)
BB4	3fff:**U**00:200:4::/64	4	Ingress PE (Customer-2)

> **i How to read a locator in this plan**
>
> All four locators share the same 40-bit locator block, 3fff:**U**00:02, which is what makes them one SRv6 domain. The 24 bits that follow are the node ID, and in this plan they are simply 1, 2, 3 and 4, matching the router number. The remaining 16 bits of each SID are the function, which BGP allocates automatically per VRF in Activity 4. Note that the whole locator range sits in 3fff:**U**00:200::/48, deliberately clear of the 3fff:**U**00:0::/48 block used for backbone links and loopbacks, so the two categories are distinguishable from the prefix alone.

Part A: Enable SRv6 in IS-IS

Before configuring anything, confirm no SRv6 or VRF configuration exists yet on any of the four backbone routers using the following commands below:

1. "**show segment-routing srv6 locator**" command in the vtysh shell on the router
2. "**ip link show type vrf**" command in the Linux shell on router

You should see that no locators are listed, and no CUST-1 or CUST-2 VRF devices are present. Everything you build in this activity starts from a clean slate on top of the IS-IS underlay and the BGP unicast and iBGP mesh from Activities 2 and 3.

Here are the outputs on router BB1:

BB1

```
---------------------------- FRR's vtysh shell ----------------------------

u01-bb1# show segment-routing srv6 locator
Locator:
Name                ID      Prefix                   Status
------------------- ------- ------------------------ -------

u01-bb1#


------------------------------ Linux Shell ------------------------------

u01-bb1:/# ip link show type vrf
u01-bb1:/#
```

Each router needs its own SRv6 locator, flooded by IS-IS, before BGP has anything to auto-allocate VPN SIDs from.

Step 4.1: Configure SRv6 on BB1

BB1

```
vtysh
configure terminal
!
segment-routing
 srv6
  locators
   locator MAIN
    prefix 3fff:U00:200:1::/64
    format uncompressed-f4024
   exit
  exit
 exit
exit
!
router isis CORE
 segment-routing srv6
  locator MAIN
 exit
exit
!
end
```

Command	Why You Are Typing It
locator MAIN	Names this locator so it can be referenced elsewhere: in the router isis CORE block below, and later in sid vpn export auto in Part B. A router can hold more than one locator, so FRR needs a name to know which one a given command means.
prefix 3fff:**U**00:200:1::/64	Defines the address block this router's SRv6 SIDs are carved out of. This is the same prefix you registered in phpIPAM in Activity 1, which is the point of having registered it: the locator is real addressing and belongs in the plan. The final digit is the node ID, so BB1 uses :1, BB2 uses :2, and so on.
format uncompressed-f4024	Selects the uncompressed SRv6 SID format from RFC 8986. The name encodes the bit split: 40 bits of locator block, 24 bits of node ID. With a 16-bit function on top, each SID is a full 128-bit address and the locator itself is a /64. This replaces the block-len, node-len and func-bits arguments used by older FRR syntax, and it is the alternative to the compressed usid-f3216 and usid-f4816 formats that the same command offers.
router isis CORE / segment-routing srv6 / locator MAIN	This is the step that actually gets the locator flooded. Defining a locator under segment-routing alone only creates it locally; without telling the IS-IS process to advertise it, no other router in the area would learn this router's SIDs exist.

Step 4.2: Configure SRv6 on BB2

BB2

```
vtysh
configure terminal
!
segment-routing
 srv6
  locators
   locator MAIN
    prefix 3fff:U00:200:2::/64
    format uncompressed-f4024
   exit
  exit
 exit
exit
!
router isis CORE
 segment-routing srv6
  locator MAIN
 exit
exit
!
end
```

Step 4.3: Configure SRv6 on BB3

BB3

```
vtysh
configure terminal
!
segment-routing
 srv6
  locators
   locator MAIN
    prefix 3fff:U00:200:3::/64
    format uncompressed-f4024
   exit
  exit
 exit
exit
!
router isis CORE
 segment-routing srv6
  locator MAIN
 exit
exit
!
end
```

Step 4.4: Configure SRv6 on BB4

BB4

```
vtysh
configure terminal
!
segment-routing
 srv6
  locators
   locator MAIN
    prefix 3fff:U00:200:4::/64
    format uncompressed-f4024
   exit
  exit
 exit
exit
!
router isis CORE
 segment-routing srv6
  locator MAIN
 exit
exit
!
end
```

> ⚠ **Before moving on, confirm all four locators are different**
>
> Each of the four routers must have its own distinct locator prefix, differing in the node digit: :1 for BB1, :2 for BB2, :3 for BB3, :4 for BB4, exactly as listed in the SRv6 Locators table in the

Quick Reference. Run show segment-routing srv6 locator on all four in sequence and compare the prefixes against that table before continuing.

i Re-entering prefix replaces the value, it does not add a second one

If you need to correct a locator prefix, simply enter the new prefix line under the same locator MAIN block. FRR replaces the existing value rather than adding a second prefix to the locator. You do not need to delete the locator first, and you do not need to re-enter the format line or the router isis CORE reference, both of which survive the change. This is the same behaviour you will meet again with rt vpn import in Part D.

Step 4.5: Verify SRv6 Locator Advertisement

First confirm the locator exists locally and is Up on each router.

BB1

```
# show segment-routing srv6 locator
```

```
u01-bb1# show segment-routing srv6 locator
Locator:
Name                ID      Prefix                   Status
------------------- ------- ------------------------ -------
MAIN                     1 3fff:100:200:1::/64      Up

u01-bb1#
```

Then confirm BB1 has learnt the other router's SRv6 locators through IS-IS.

```
# show ipv6 route isis
```

You should see the **SRv6 locators** of the other routers in the backbone as ordinary IS-IS routes, plus this router's own **adjacency SIDs**, which appear as directly connected /128 entries carrying a **seg6local** action.

BB1

```
u01-bb1# show ipv6 route isis
Codes: K - kernel route, C - connected, L - local, S - static,
       R - RIPng, O - OSPFv3, I - IS-IS, B - BGP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric, t - Table-Direct,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

IPv6 unicast VRF default:
I>* 3fff:100::2/128 [115/20] via fe80::a8c1:abff:fe02:705a, eth2, weight 1, 05:00:05
I>* 3fff:100::3/128 [115/20] via fe80::a8c1:abff:fe45:8f56, eth3, weight 1, 02:33:13
I>* 3fff:100::4/128 [115/20] via fe80::a8c1:abff:fe98:49af, eth4, weight 1, 02:32:53
I>* 3fff:100:0:23::/64 [115/20] via fe80::a8c1:abff:fe02:705a, eth2, weight 1, 02:33:13
  *                            via fe80::a8c1:abff:fe45:8f56, eth3, weight 1, 02:33:13
I>* 3fff:100:0:24::/64 [115/20] via fe80::a8c1:abff:fe02:705a, eth2, weight 1, 02:32:53
  *                            via fe80::a8c1:abff:fe98:49af, eth4, weight 1, 02:32:53
```

```
I>* 3fff:100:0:34::/64 [115/20] via fe80::a8c1:abff:fe45:8f56, eth3, weight 1, 02:32:53
  *                            via fe80::a8c1:abff:fe98:49af, eth4, weight 1, 02:32:53
I>* 3fff:100:200:1:41::/128 [115/0] is directly connected, eth4, seg6local End.X nh6 fe80::a8c1:abff:fe98:49af, weight 1,
00:11:58
I>* 3fff:100:200:1:42::/128 [115/0] is directly connected, eth3, seg6local End.X nh6 fe80::a8c1:abff:fe45:8f56, weight 1,
00:11:58
I>* 3fff:100:200:1:43::/128 [115/0] is directly connected, eth2, seg6local End.X nh6 fe80::a8c1:abff:fe02:705a, weight 1,
00:11:58
I>* 3fff:100:200:2::/64 [115/10] via fe80::a8c1:abff:fe02:705a, eth2, weight 1, 00:11:00
I>* 3fff:100:200:3::/64 [115/10] via fe80::a8c1:abff:fe45:8f56, eth3, weight 1, 00:04:11
I>* 3fff:100:200:4::/64 [115/10] via fe80::a8c1:abff:fe98:49af, eth4, weight 1, 00:02:38
u01-bb1#
```

You can also inspect what the kernel has actually programmed, from the Linux shell on BB1.

BB1

```
# ip -6 route show | grep seg6local
```

```
u01-bb1:/# ip -6 route show | grep seg6local
3fff:100:200:1:41:: nhid 54  encap seg6local action End.X nh6 fe80::a8c1:abff:fe98:49af dev eth4 proto isis
metric 20 pref medium
3fff:100:200:1:42:: nhid 55  encap seg6local action End.X nh6 fe80::a8c1:abff:fe45:8f56 dev eth3 proto isis
metric 20 pref medium
3fff:100:200:1:43:: nhid 56  encap seg6local action End.X nh6 fe80::a8c1:abff:fe02:705a dev eth2 proto isis
metric 20 pref medium
u01-bb1:/#
```

This shows the SRv6 adjacency SIDs the kernel has programmed on this router, one per IS-IS neighbour link.

Field	What It Means
action End.X	SRv6's endpoint-with-cross-connect behaviour. A packet arriving with one of these addresses as its destination tells this router to decapsulate and forward specifically out this one link, to this one neighbour: a forced next hop rather than a general routing lookup.
nh6 fe80::...	The link-local address of that neighbour, which is where the packet gets forwarded.
proto isis	These were not manually configured. IS-IS auto-generated and installed them, one per adjacency, the moment the locator was advertised.
metric	The IS-IS cost associated with reaching that neighbour.
(no flavors field)	Under the uncompressed format there is no compression flavour to apply, so no flavors attribute appears. If you have seen output elsewhere carrying "flavors next-csid lblen 32 nflen 16", that is a uSID build, not this one.

i Checkpoint

The underlay now carries SRv6 locators, but no service is riding on them yet; it is still just IPv6 reachability with extra addressing. The VPN services come next.

Part B: Connect Customer-1's private service in the Data Centre (Storage)

BB3 is the ingress PE for Customer-1. It receives IPv6 traffic from ER, encapsulates it with **BB1's End.DT6 SID** as the outer destination via **SRv6 H.Encaps**, and forwards it across the backbone. BB1, as egress PE, decapsulates and delivers it into the Data Centre toward Storage.

> **i What an End.DT6 SID is**
>
> An End.DT6 SID is a network instruction encoded as an address. It tells an edge router to remove the outer IPv6 header, look up the inner destination in one specific IPv6 routing table, and deliver the packet from there. The DT6 part is the behaviour: Decapsulate and lookup in an IPv6 Table. Each VRF on a PE gets its own End.DT6 SID, which is precisely how the router knows which customer a decapsulated packet belongs to.

Quick Reference Guide: Data Centre VLANs

VLAN	Prefix (/64)	Purpose	Server	Server Addr
VLAN 10	3fff:**U**00:d:10::/64	CUST-1 IPv6 L3VPN	Storage	3fff:**U**00:d:10::a
VLAN 21	3fff:**U**00:d:21::/64	CUST-2 IPv6 L3VPN	WEB	3fff:**U**00:d:21::b
VLAN 100	3fff:**U**00:0:e::/64	Shared services (SVC-DNS VRF, extranet)	DNS	3fff:**U**00:0:e::e

Customer-1 VPN Service Addressing (PI space)

Customer-1 prefix: 2001:db8:**U**23::/48, which covers Host-A's SLAAC address in 2001:db8:**U**23:0::/64.

Router	Interface	Address	VRF
BB3 (ingress PE)	eth0 -> ER	3fff:**U**00:0:36::3/64	CUST-1
BB1 (egress PE)	eth0.10 -> DC-Switch VLAN 10	3fff:**U**00:d:10::1/64	CUST-1
ER	eth1 -> BB3	3fff:**U**00:0:36::6/64	global
Storage server	eth0	3fff:**U**00:d:10::a/64	Data Centre

Customer-2 VPN Service Addressing (delegated space)

Customer-2 prefix: 3fff:**U**00:21::/48, which covers Host-B's SLAAC address in 3fff:**U**00:21:0::/64.

Router	Interface	Address	VRF
BB4 (ingress PE)	eth0 -> CPE	3fff:**U**00:0:49::4/64	CUST-2
BB1 (egress PE)	eth0.21 -> DC-Switch VLAN 21	3fff:**U**00:d:21::1/64	CUST-2
CPE	eth1 -> BB4	3fff:**U**00:0:49::9/64	global
WEB server	eth0	3fff:**U**00:d:21::b/64	Data Centre

Step 4.6: Create the Linux VRF on BB3

FRR needs a real kernel-level VRF to bind the customer-facing interface to. Create it in the Linux shell before touching BGP.

BB3 - Linux Shell

```
ip link add CUST-1 type vrf table 10
ip link set CUST-1 up
ip link set eth0 vrf CUST-1
ip link set eth0 up
```

Step 4.7: Configure BGP and the CUST-1 VRF on BB3 (Ingress PE)

Before any per-VRF policy can carry a route anywhere, the underlying iBGP session toward BB1 needs to be activated for the VPNv6 address family, and **BGP needs to know which SRv6 locator to allocate VPN SIDs from**. This is a global setting on the router bgp **U**00 process, not something that lives inside the per-VRF instance, so it is a one-time addition rather than something you repeat per customer.

Quick Reference Guide: BGP VPNv6 Parameters

Parameter	CUST-1	CUST-2	SVC-DNS
AS Number	**U**00	**U**00	**U**00
VRF name	CUST-1	CUST-2	SVC-DNS
Routing table ID	10	21	100
Route Distinguisher	**U**00:10	**U**00:21	**U**00:100

Parameter	CUST-1	CUST-2	SVC-DNS
Route Target (own)	U00:10	U00:21	U00:100
Additional RT imported	U00:100 (extranet)	U00:100 (extranet)	U00:10, U00:21 (extranet)
PE loopback	3fff:U00::3 (BB3, ingress) / 3fff:U00::1 (BB1, egress)	3fff:U00::4 (BB4, ingress) / 3fff:U00::1 (BB1, egress)	3fff:U00::1 (BB1 only)

BB3

```
vtysh
configure terminal
router bgp U00
 segment-routing srv6
  locator MAIN
 exit
 address-family ipv6 vpn
  neighbor 3fff:U00::1 activate
  neighbor 3fff:U00::1 send-community both
 exit-address-family
end
```

Command	Why You Are Typing It
segment-routing srv6 / locator MAIN (under the global router bgp U00)	This looks exactly like the block you added to router isis CORE in Part A, but it is separate. In FRR, IS-IS and BGP are different daemons, and each needs its own reference to the locator. **The IS-IS daemon uses its copy to advertise locator reachability to neighbours. The BGP daemon uses its copy to allocate and advertise VPN SIDs.**
address-family ipv6 vpn (under the global router bgp U00)	Without this, show bgp ipv6 vpn summary will show zero neighbors. You are about to configure import and export rules for VRF traffic, and those rules cannot work unless the iBGP session is activated for VPNv6 first.
neighbor … activate	Turns on the exchange of VPNv6 routing information with BB1, the egress PE this VRF must connect to. Activation is per neighbor and per address family, and it does not affect the ordinary IPv6 unicast routing you configured in Activity 3.
neighbor … send-community both	Route Targets are carried as BGP extended communities. Without this command the router will not send them, so BB1 would receive the routes untagged, fail to match them against its import rules, and discard them.

BB3

```
vtysh
configure terminal
router bgp U00 vrf CUST-1
 address-family ipv6 unicast
  redistribute connected
  sid vpn export auto
  rd vpn export U00:10
  rt vpn both U00:10
  export vpn
  import vpn
 exit-address-family
end
```

Command	Why You Are Typing It
router bgp **U**00 vrf CUST-1	Creates a VRF-scoped BGP instance bound to the Linux VRF you just created. This is where Customer-1's routes live, separate from the global table's BGP instance.
redistribute connected	Pulls BB3's directly connected CUST-1 interface, the link to ER, into BGP so it can be exported into the VPN.
sid vpn export auto	Allocates an **End.DT6 SID** for this VRF automatically out of BB3's locator, using the 16-bit function field. **Other routers use this SID as the outer destination address when sending traffic destined for this specific VRF**.
rd vpn export **U**00:10	A Route Distinguisher prefixes Customer-1's ordinary IPv6 prefixes with a unique value inside BGP, so that two customers using identical address space can't be confused with one another.
rt vpn both **U**00:10	A Route Target is the tag that controls which VRFs may accept a route. Using both sets the same value for export and import, which is what keeps Customer-1 talking only to itself at this stage.
export vpn / import vpn	These two lines turn on the actual movement of routes between this local VRF and the global VPNv6 table. Without them the RD and RT settings above are configured but inert.

Step 4.8: Add a Static Route for Customer-1's LAN on BB3

ER advertises 2001:db8:**U**23:0::/64 to Host-A via SLAAC, but BB3 still needs a route toward it through ER.

BB3

```
configure terminal
ipv6 route 2001:db8:U23::/48 3fff:U00:0:36::6 vrf CUST-1
router bgp U00 vrf CUST-1
 address-family ipv6 unicast
  redistribute static
 exit-address-family
end
```

Step 4.9: Create the Linux VRF and Configure BGP on BB1 (Egress PE)

BB1 - Linux Shell

```
ip link add CUST-1 type vrf table 10
ip link set CUST-1 up
ip link set eth0.10 vrf CUST-1
ip link set eth0.10 up
```

As on **BB3** in **Step 4.7**, the global router bgp **U**00 process needs its own locator reference for the BGP daemon, and the iBGP session toward BB3 needs to be activated for VPNv6 before any per-VRF policy can carry a route across it.

BB1

```
vtysh
configure terminal
router bgp U00
 segment-routing srv6
  locator MAIN
 exit
 address-family ipv6 vpn
  neighbor 3fff:U00::3 activate
  neighbor 3fff:U00::3 send-community both
 exit-address-family
end
```

BB1

```
vtysh
configure terminal
router bgp U00 vrf CUST-1
 address-family ipv6 unicast
  redistribute connected
  sid vpn export auto
  rd vpn export U00:10
  rt vpn both U00:10
  export vpn
  import vpn
 exit-address-family
end
```

Part C: Connect Customer-2's Private Service in the Data Centre (WEB)

BB4 is the ingress PE for Customer-2. The steps mirror Part B exactly, using table 21, RD and RT **U**00:21, and BB1's VLAN 21 sub-interface instead of VLAN 10. Every command here does exactly what its Part B counterpart did (see the command tables under Step 4.7); only the table number, RD/RT value, and VLAN change.

Step 4.10: Create the Linux VRF and Configure BGP on BB4 (Ingress PE)

BB4 - Linux Shell

```
ip link add CUST-2 type vrf table 21
ip link set CUST-2 up
ip link set eth0 vrf CUST-2
ip link set eth0 up
```

BB4

```
vtysh
configure terminal
router bgp U00
 segment-routing srv6
  locator MAIN
 exit
 address-family ipv6 vpn
  neighbor 3fff:U00::1 activate
  neighbor 3fff:U00::1 send-community both
 exit-address-family
end
```

BB4

```
vtysh
configure terminal
router bgp U00 vrf CUST-2
 address-family ipv6 unicast
  redistribute connected
  sid vpn export auto
  rd vpn export U00:21
  rt vpn both U00:21
  export vpn
  import vpn
 exit-address-family
end
```

Step 4.11: Add a Static Route for Customer-2's LAN on BB4

BB4

```
configure terminal
ipv6 route 3fff:U00:21::/48 3fff:U00:0:49::9 vrf CUST-2
router bgp U00 vrf CUST-2
 address-family ipv6 unicast
  redistribute static
 exit-address-family
end
```

Step 4.12: Create the Linux VRF and Configure BGP on BB1 (Egress PE, Additive)

BB1 - Linux Shell

```
ip link add CUST-2 type vrf table 21
ip link set CUST-2 up
ip link set eth0.21 vrf CUST-2
ip link set eth0.21 up
```

BB1 already has its own locator referenced under router bgp **U**00 from Step 4.9, so that part does not need to be repeated. It still needs the iBGP session toward BB4 activated for VPNv6, which Step 4.9 did not cover since that step only activated the session toward BB3.

BB1

```
vtysh
configure terminal
router bgp U00
 address-family ipv6 vpn
  neighbor 3fff:U00::4 activate
  neighbor 3fff:U00::4 send-community both
 exit-address-family
end
```

BB1

```
vtysh
configure terminal
router bgp U00 vrf CUST-2
 address-family ipv6 unicast
  redistribute connected
  sid vpn export auto
  rd vpn export U00:21
  rt vpn both U00:21
  export vpn
  import vpn
 exit-address-family
end
```

Part D: Give the Shared DNS Server its Own Small VRF

DNS lives on VLAN 100 (3fff:**U**00:0:e::/64), in its own small VRF, SVC-DNS, table 100, RD/RT **U**00:100, that you'll build in this activity. It isn't inside either customer's VPN and isn't in the global table either. It's a third VRF connected to both customers through an Extranet VPN pattern: SVC-DNS imports both customers' Route Targets, and both customers import SVC-DNS's Route Target, without either customer importing the other's.

Quick reference : BGP VPNv6 Parameters SVC-DNS

Parameter	SVC-DNS
AS Number	**U**00
VRF name	SVC-DNS
Routing table ID	100
Route Distinguisher	**U**00:100
Route Target (own)	**U**00:100
Additional RT imported	**U**00:10, **U**00:21 (extranet)
PE loopback	3fff:**U**00::1 (BB1 only)

Step 4.13: Create the SVC-DNS VRF on BB1

DNS was originally left in the global table on VLAN 100. It needs to move into its own VRF instead.

BB1 - Linux Shell

```
ip link add SVC-DNS type vrf table 100
ip link set SVC-DNS up
ip link set eth0.100 vrf SVC-DNS
ip link set eth0.100 up
```

BB1

```
vtysh
configure terminal
router bgp U00 vrf SVC-DNS
 address-family ipv6 unicast
  redistribute connected
  sid vpn export auto
  rd vpn export U00:100
  rt vpn both U00:100
  rt vpn import U00:10 U00:21
  export vpn
```

```
  import vpn
 exit-address-family
end
```

Command	Why You Are Typing It
rt vpn both U00:100	Tells the SVC-DNS VRF to export its own routes tagged U00:100 and to accept incoming routes carrying that same tag. This is the same standalone arrangement used for every other VRF in this lab.
rt vpn import U00:10 U00:21	Turns this VRF into an extranet hub rather than an isolated VPN. Without this line SVC-DNS has no route back to either customer, so a query would reach the DNS server correctly but the reply would be dropped at BB1. A hub must hold routes back to every spoke.

Step 4.14: Import the Shared RT Into Each Customer VRF on BB1

Both of BB1's existing VRF instances need one added line each; nothing else about them changes.

BB1

```
vtysh
configure terminal
router bgp U00 vrf CUST-1
 address-family ipv6 unicast
  rt vpn import U00:10 U00:100
 exit-address-family
exit
router bgp U00 vrf CUST-2
 address-family ipv6 unicast
  rt vpn import U00:21 U00:100
 exit-address-family
end
```

> ⚠ **rt vpn import replaces the list, it does not add to it**
>
> This is the single most consequential FRR behaviour in this entire activity, and it is not obvious from the syntax. CUST-1 already has an import policy from Step 4.7 (rt vpn both U00:10, which sets import to U00:10). Running rt vpn import U00:100 on its own here would not add U00:100 to that list; it silently replaces the whole list with just U00:100.
>
> If you do that, CUST-1's own route to Storage stops being imported at all, while DNS starts working, which is exactly backwards from what you would expect to break. The fix is always to list every RT a VRF needs to import on a single rt vpn import line, its own RT included, never as a separate later command. This applies every time you touch an import list again.

> ⚠ **Check the VRF instance exists before you add an import line to it**
>
> Before adding an import line, run show running-config on the router and compare the target VRF instance against a working one. A correctly built instance has seven lines under address-family ipv6 unicast; a stub created this way has one.

Step 4.15: Import the Shared RT on Each Ingress PE Too

The same replace-not-add rule from Step 4.14 applies here too, so restate the full import list including the VRF's own RT.

BB3

```
vtysh
configure terminal
router bgp U00 vrf CUST-1
 address-family ipv6 unicast
  rt vpn import U00:10 U00:100
 exit-address-family
end
```

BB4

```
vtysh
configure terminal
router bgp U00 vrf CUST-2
 address-family ipv6 unicast
  rt vpn import U00:21 U00:100
 exit-address-family
end
```

Because the new configuration overwrites the old import list rather than adding to it, and because changing policy does not retroactively recover routes the router already rejected and discarded, you must force BGP to re-evaluate all inbound routes.

On BB3 and BB4:

```
clear bgp ipv6 vpn 3fff:U00::1 in
```

On BB1:

```
clear bgp ipv6 vpn 3fff:U00::3 in
clear bgp ipv6 vpn 3fff:U00::4 in
```

BB3

```
u01-bb3# show ipv6 route vrf CUST-1
Codes: K - kernel route, C - connected, L - local, S - static,
       R - RIPng, O - OSPFv3, I - IS-IS, B - BGP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric, t - Table-Direct,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

IPv6 unicast VRF CUST-1:
S>* 2001:db8:123::/48 [1/0] via 3fff:100:0:36::6, eth0, weight 1, 00:35:14
B>  3fff:100:0:e::/64 [200/0] via 3fff:100::1 (vrf default) (recursive), label 1120, seg6
3fff:100:200:1:46::, weight 1, 00:03:32
  *                          via fe80::a8c1:abff:feb6:8c6b, eth1 (vrf default), label 1120, seg6
3fff:100:200:1:46::, weight 1, 00:03:32
C>* 3fff:100:0:36::/64 is directly connected, eth0, weight 1, 00:45:55
L>* 3fff:100:0:36::3/128 is directly connected, eth0, weight 1, 00:45:55
B>  3fff:100:d:10::/64 [200/0] via 3fff:100::1 (vrf default) (recursive), label 1088, seg6
3fff:100:200:1:44::, weight 1, 00:03:32
  *                           via fe80::a8c1:abff:feb6:8c6b, eth1 (vrf default), label 1088, seg6
3fff:100:200:1:44::, weight 1, 00:03:32
C>* fe80::/64 is directly connected, eth0, weight 1, 00:45:55
u01-bb3#
```

Prefix	Code	Why it has that code
2001:db8:**U**23::/48	S>*	Customer-1's LAN, the static route via ER you configured in Step 4.8. Local to BB3, so it never travels over VPNv6.
3fff:**U**00:0:e::/64	B>	DNS, learned from BB1 through the extranet import. Note the seg6 SID ending :46, which is the SVC-DNS End.DT6 SID on BB1.
3fff:**U**00:d:10::/64	B>	Storage, learned from BB1 through CUST-1's own Route Target. Its seg6 SID ends :44, a different function on the same BB1 locator.
3fff:**U**00:0:36::/64	C>*	The local link to ER, an ordinary connected route inside the VRF.

Part E: Verify BGP VPNv6 Sessions and SID Allocation

Step 4.16: Verify BGP VPNv6 Sessions and SID Allocation

The VRFs are built, so now confirm the control plane is actually carrying them. There are two things to check, and it is worth keeping them apart: first, that the VPNv6 sessions between the PEs are up and exchanging prefixes, then that the routes crossing those sessions carry the SRv6 SIDs the far end needs.

Start with the sessions. Run **show bgp ipv6 vpn summary** on **BB1**, **BB3** and **BB4** — the three routers holding VRFs. BB2 has none, so it has no VPNv6 sessions and isn't part of this check.

BB1

```
# show bgp ipv6 vpn summary
```

```
u01-bb1# show bgp ipv6 vpn summary
BGP router identifier 10.1.1.1, local AS number 100 VRF default vrf-id 0
BGP table version 0
RIB entries 5, using 640 bytes of memory
Peers 2, using 33 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
3fff:100::3     4        100       150       162        3    0    0 00:42:21            2        3
FRRouting/10.5.0_git
3fff:100::4     4        100       148       160        3    0    0 00:29:07            2        3
FRRouting/10.5.0_git

Total number of neighbors 2
u01-bb1#
```

BB4

```
u01-bb4# show bgp ipv6 vpn summary
BGP router identifier 10.1.4.4, local AS number 100 VRF default vrf-id 0
BGP table version 0
RIB entries 5, using 640 bytes of memory
Peers 1, using 17 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
3fff:100::1     4        100       162       153        3    0    0 00:34:46            3        2
FRRouting/10.5.0_git

Total number of neighbors 1
u01-bb4#
```

BB1 is the egress PE for both customers, so it shows two neighbours, BB3 and BB4. BB3 and BB4 each peer only with BB1 for VPNv6, so they show one. In every case, the session should be Established with non-zero counts under both **State/PfxRcd** and **PfxSnt**. A neighbour sitting in Active or Idle means Step 4.7, 4.9 or 4.12 was missed on one end.

Now look at the VPNv6 table itself. Note that this command, with no VRF specified, shows the router's global BGP VPNv6 table. It shows everything it received from any peer over the VPNv6 address family, including routes tagged with the other customer's Route Distinguisher, rather than what is actually usable in any one VRF.

BB3

```
# show bgp ipv6 vpn
```

```
u01-bb3# show bgp ipv6 vpn
BGP table version is 3, local router ID is 10.1.3.3, vrf id 0
Default local pref 100, local AS 100
Status codes:  s suppressed, d damped, h history, u unsorted, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
Route Distinguisher: 100:10
 *>  2001:db8:123::/48
                    3fff:100:0:36::6@2<
                                                0         32768 ?
    UN=3fff:100:0:36::6 EC{100:10} label=1088 sid=3fff:100:200:3:: sid_structure=[40,24,16,0]
type=bgp, subtype=5
 *>  3fff:100:0:36::/64
                    ::@2<                       0         32768 ?
    UN=:: EC{100:10} label=1088 sid=3fff:100:200:3:: sid_structure=[40,24,16,0] type=bgp,
subtype=5
 *>i 3fff:100:d:10::/64
                    3fff:100::1                 0    100      0 ?
    UN=3fff:100::1 EC{100:10} label=1088 sid=3fff:100:200:1:: sid_structure=[40,24,16,0]
type=bgp, subtype=0
Route Distinguisher: 100:21
 *>i 3fff:100:d:21::/64
                    3fff:100::1                 0    100      0 ?
    UN=3fff:100::1 EC{100:21} label=1104 sid=3fff:100:200:1:: sid_structure=[40,24,16,0]
type=bgp, subtype=0
Route Distinguisher: 100:100
 *>i 3fff:100:0:e::/64
                    3fff:100::1                 0    100      0 ?
    UN=3fff:100::1 EC{100:100} label=1120 sid=3fff:100:200:1:: sid_structure=[40,24,16,0]
type=bgp, subtype=0

Displayed 5 routes and 5 total paths
u01-bb3#
```

Step 4.17: Verify Kernel SRv6 Programming

Everything up to this point has been the control plane: BGP sessions, BGP tables, and what the routers have agreed among themselves. This step checks the other half: what the kernel (data plane) actually installed and will forward on. The two are separate events, and a route can be selected as best in a BGP table without ever reaching the forwarding table.

Each VRF has its own kernel routing table, numbered to match the table ID you gave it when you created the VRF: table 10 for CUST-1, 21 for CUST-2, 100 for SVC-DNS. Check each router that holds that VRF and confirm two things: that every prefix the VPN should carry is present, and that prefixes belonging to a remote router carry **encap seg6** with a SID in the segment list, while locally attached ones do not.

On Linux Shell of BB1:

```
# ip -6 route show table 10
# ip -6 route show table 21
# ip -6 route show table 100
```

On Linux Shell of BB3:

```
# ip -6 route show table 10
```

On Linux Shell of BB4:

```
# ip -6 route show table 21
```

BB1 and BB3 table 10 are shown as a worked example; run the rest yourself.

BB1

```
u01-bb1:/# ip -6 route show table 10
2001:db8:123::/48 nhid 68  encap seg6 mode encap segs 1 [ 3fff:100:200:3:44:: ] via
fe80::a8c1:abff:fe45:8f56 dev eth3 proto bgp metric 20 pref medium
3fff:100:0:e::/64 nhid 94 dev SVC-DNS proto bgp metric 20 pref medium
3fff:100:0:36::/64 nhid 68  encap seg6 mode encap segs 1 [ 3fff:100:200:3:44:: ] via
fe80::a8c1:abff:fe45:8f56 dev eth3 proto bgp metric 20 pref medium
anycast 3fff:100:d:10:: dev eth0.10 proto kernel metric 0 pref medium
local 3fff:100:d:10::1 dev eth0.10 proto kernel metric 0 pref medium
3fff:100:d:10::/64 dev eth0.10 proto kernel metric 256 pref medium
anycast fe80:: dev eth0.10 proto kernel metric 0 pref medium
local fe80::a8c1:abff:fe63:8a11 dev eth0.10 proto kernel metric 0 pref medium
fe80::/64 dev eth0.10 proto kernel metric 256 pref medium
multicast ff00::/8 dev eth0.10 proto kernel metric 256 pref medium
u01-bb1:/#
```

BB3

```
u01-bb3:/# ip -6 route show table 10
2001:db8:123::/48 nhid 53 via 3fff:100:0:36::6 dev eth0 proto 196 metric 20 pref medium
3fff:100:0:e::/64 nhid 62  encap seg6 mode encap segs 1 [ 3fff:100:200:1:46:: ] via
fe80::a8c1:abff:feb6:8c6b dev eth1 proto bgp metric 20 pref medium
anycast 3fff:100:0:36:: dev eth0 proto kernel metric 0 pref medium
local 3fff:100:0:36::3 dev eth0 proto kernel metric 0 pref medium
3fff:100:0:36::/64 dev eth0 proto kernel metric 256 pref medium
3fff:100:d:10::/64 nhid 58  encap seg6 mode encap segs 1 [ 3fff:100:200:1:44:: ] via
fe80::a8c1:abff:feb6:8c6b dev eth1 proto bgp metric 20 pref medium
anycast fe80:: dev eth0 proto kernel metric 0 pref medium
local fe80::a8c1:abff:fed8:fc1d dev eth0 proto kernel metric 0 pref medium
fe80::/64 dev eth0 proto kernel metric 256 pref medium
multicast ff00::/8 dev eth0 proto kernel metric 256 pref medium
u01-bb3:/#
```

> **i One line in the BB1 output is worth a second look**
>
> DNS appears as 3fff:U00:0:e::/64 … dev SVC-DNS proto bgp with no seg6 encapsulation at all, unlike every other imported prefix here. That is correct: on BB1 the DNS segment is local, sitting in the SVC-DNS VRF on this same router, so the extranet import gives CUST-1 a

route that points straight at the neighbouring VRF device rather than a tunnel. Encapsulation only appears when the destination genuinely lives on a different router.

Each router only encapsulates a packet when the destination sits on the other router. If the destination is local, it delivers the packet directly, with no encapsulation needed. So on BB3, Storage and DNS are remote (on BB1), so BB3 encapsulates traffic toward them, while Customer-1's LAN is local to BB3 and needs no encapsulation. On BB1 the reverse applies. In short: encapsulate to send it away; do not encapsulate if it is already home.

The End.DT6 SIDs themselves, the actual decapsulation instructions those VRF routes point toward, do not live inside either of the tables above. They sit in the global seg6local namespace instead. See all of them together on BB1:

In the Linux shell of BB1:

```
# ip -6 route show | grep seg6local
```

```
u01-bb1:/# ip -6 route show | grep seg6local
3fff:100:200:1:41:: nhid 54  encap seg6local action End.X nh6 fe80::a8c1:abff:fe98:49af dev eth4 proto isis
metric 20 pref medium
3fff:100:200:1:42:: nhid 55  encap seg6local action End.X nh6 fe80::a8c1:abff:fe45:8f56 dev eth3 proto isis
metric 20 pref medium
3fff:100:200:1:43:: nhid 56  encap seg6local action End.X nh6 fe80::a8c1:abff:fe02:705a dev eth2 proto isis
metric 20 pref medium
3fff:100:200:1:44:: nhid 65  encap seg6local action End.DT6 table 10 dev CUST-1 proto bgp metric 20 pref
medium
3fff:100:200:1:45:: nhid 75  encap seg6local action End.DT6 table 21 dev CUST-2 proto bgp metric 20 pref
medium
3fff:100:200:1:46:: nhid 85  encap seg6local action End.DT6 table 100 dev SVC-DNS proto bgp metric 20 pref
medium
u01-bb1:/#
```

i What you're looking at

This is a list of special addresses that tell BB1 what to do with packets arriving to them. Each line is one rule, and there are two kinds.

The End.X entries mean "send this packet out to one specific neighbour", with one rule for each router BB1 is directly connected to and has an adjacency with. The End.DT6 entries mean "decapsulate this packet and look up the inner destination in one specific customer table", with one rule for each service: Customer-1 (table 10), Customer-2 (table 21) and shared DNS (table 100).

All of these addresses share the same locator prefix, which simply means they belong to BB1. What distinguishes them is the 16-bit function at the end, allocated automatically. Every line should have a distinct full address; two rules sharing an identical address would be a genuine fault.

Step 4.18: Watch an SRv6-Encapsulated Packet Cross the Core

Everything so far has confirmed the control plane and the kernel state. This step proves it by watching an actual packet. In the Linux shell of BB1, capture on eth3, the direct link to BB3, while generating traffic from Host-A to Storage.

```
BB1# tcpdump -i eth3 -n -v ip6
```

From another window, generate traffic from Host-A to Storage while the capture is running.

```
ping6 -c 5 3fff:U00:d:10::a
```

In the capture you should see two nested IPv6 headers, not one.

BB1

```
u01-bb1:/# tcpdump -i eth3 -n -v ip6
tcpdump: listening on eth3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
18:02:38.028953 IP6 (flowlabel 0x01d29, hlim 62, next-header Routing (43) payload length: 128)
3fff:100:0:13::3 > 3fff:100:200:1:44::: RT6 (len=2, type=4, segleft=0, last-entry=0, flags=0x0,
tag=0, [0]3fff:100:200:1:44::) IP6 (flowlabel 0x01d29, hlim 63, next-header ICMPv6 (58) payload
length: 64) 2001:db8:123:0:dd09:33a9:1f8a:9402 > 3fff:100:d:10::a: [icmp6 sum ok] ICMP6, echo
request, id 9, seq 1
```

```
18:02:38.029090 IP6 (flowlabel 0x5dd8c, hlim 63, next-header Routing (43) payload length: 128)
3fff:100:0:13::1 > 3fff:100:200:3:44::: RT6 (len=2, type=4, segleft=0, last-entry=0, flags=0x0,
tag=0, [0]3fff:100:200:3:44::) IP6 (flowlabel 0x5dd8c, hlim 64, next-header ICMPv6 (58) payload
length: 64) 3fff:100:d:10::a > 2001:db8:123:0:dd09:33a9:1f8a:9402: [icmp6 sum ok] ICMP6, echo
reply, id 9, seq 1
```

> **i How to read the capture**
>
> The outer header is the tunnel. Its source is the encapsulating PE's address on that link and its destination is the far PE's End.DT6 SID, which is exactly one of the addresses you saw in the seg6local list in Step 4.17. The **RT6 line** is the Segment Routing Header, and segleft=0 means there are no further segments to visit, consistent with a single-segment path where the ingress PE encapsulates directly to the egress PE with nothing in between.
>
> The inner header is the customer packet, completely untouched, exactly as the CE router originally sent it.
>
> Note that the reply is also encapsulated rather than simply forwarded natively. BB1 re-encapsulates it because BB3, not BB1, is where the routing table for Host-A's site actually lives.

Now repeat the same capture on eth4, the direct link to BB4, while pinging from Host-B to WEB, and confirm the outer destination changes to reflect BB4 as ingress PE instead of BB3.

In the Linux shell of BB1:

```
# tcpdump -i eth4 -n -v ip6
```

From Host-B:

```
# ping6 -c 5 3fff:U00:d:21::b
```

> ⚠ **If you don't see two headers**
>
> A single flat IPv6 header, with no encapsulation visible, means traffic is being routed in the global table rather than forwarded through the VPN service. Go back and check that the relevant VRF has both a BGP session in Established state (Step 4.16) and a populated kernel table (Step 4.17) before retesting.

Step 4.19: End-to-End Reachability

From Host-A, ping Storage. From Host-B, ping WEB. Both should succeed.

```
# On Host-A
ping6 3fff:U00:d:10::a

# On Host-B
ping6 3fff:U00:d:21::b
```

Both customers should also now be able to reach the shared DNS server. Confirm from both Host A and Host B.

```
ping6 3fff:U00:0:e::e
```

> ⚠ **If DNS is unreachable but Storage and WEB work fine**
>
> First, isolate whether the problem is local to the Data Centre segment or somewhere in the VPN path. Ping DNS directly from inside BB1's own SVC-DNS VRF, bypassing the customer VRFs entirely, with ip vrf exec SVC-DNS ping 3fff:U00:0:e::e. If that fails too, the fault is local, meaning the DNS server itself or BB1's eth0.100 segment, and nothing in the VPN chain you just built. If it succeeds but the customer-originated ping still does not, re-check Steps 4.13 to 4.15 in order: a missing rt vpn import on any one of the three affected VRF instances produces this symptom exactly.

Step 4.20: Verify VPN Isolation

This is the step that proves the VPN is actually private, not merely working. Confirm Customer-1 cannot reach Customer-2's WEB server, and Customer-2 cannot reach Customer-1's Storage server.

```
# From Host-A (Customer-1), should fail
ping6 3fff:U00:d:21::b

# From Host-B (Customer-2), should fail
ping6 3fff:U00:d:10::a
```

> **i Both pings should time out**
>
> If either of these succeeds, one of the VRFs has the wrong Route Target and is importing the other customer's routes; go back and double-check the rt vpn both and rt vpn import commands on the affected router. Note that both customers being able to reach DNS is expected and correct, that is the extranet working as designed. This check is specifically about the two customers never reaching each other's private service.

Activity 4 Summary

- You gave each backbone router an SRv6 locator using the uncompressed RFC 8986 SID format, and flooded it through IS-IS.
- You built independent VRFs for Customer-1 and Customer-2 on their respective ingress and egress PEs, using BGP VPNv6 with unique Route Distinguishers and Route Targets.
- You delivered each customer's private Data Centre service, Storage for Customer-1 and WEB for Customer-2.
- You built a third VRF for the shared DNS service and connected it to both customers using an Extranet VPN pattern, importing each customer's Route Target into the DNS VRF and the DNS Route Target into each customer's VRF, without merging the two customers' VPNs into each other.
- You proved the two customer VPN services are fully isolated from one another, while both can still reach the shared DNS service.
- You confirmed at each stage that a route selected in a BGP table and a route installed in the forwarding plane are two separate things, and checked both.

Activity 5: Configure DNS

With both VPN services live, the last piece is making them usable by name instead of raw IPv6 addresses. You will configure the CPE to advertise the shared DNS server to Host-B via Router Advertisements, then add records on the DNS server for both Data Centre services.

Step 5.1: Check the Existing Configuration

Confirm Host-B currently has no IPv6 DNS server configured, and that the CPE has no RDNSS option set yet.

HostB

```
# more /etc/resolv.conf
```

```
u01-hostB / # more /etc/resolv.conf
# Generated by dhcpcd
# /etc/resolv.conf.head can replace this line
# /etc/resolv.conf.tail can replace this line
u01-hostB / #
```

resolv.conf on Host-B should show no IPv6 nameserver line. Host-B has a working SLAAC address and default route from earlier activities, but nothing has told it where to send DNS queries over IPv6 yet. On the CPE, confirm that the "**no ipv6 nd rdnss**" line exists under "eth0":

CPE

```
show running-config
```

```
u01-cpe# show running-config
Building configuration...

Current configuration:
!
frr version 10.5.0_git
frr defaults traditional
hostname u01-cpe
service integrated-vtysh-config
!
ipv6 route ::/0 3fff:100:0:49::4 eth1
!
interface eth0
 ipv6 address 3fff:100:21::1/64
 ipv6 nd ra-interval 20
 no ipv6 nd suppress-ra
exit
!
interface eth1
 ipv6 address 3fff:100:0:49::9/64
exit
!
end
u01-cpe#
```

Step 5.2: Configure RDNSS on CPE

Configure the RDNSS option on the CPE's host-facing interface, pointing at the shared DNS server (3fff:**U**00:0:e::e).

CPE

```
configure terminal
interface eth0
 ipv6 nd rdnss 3fff:U00:0:e::e
 no ipv6 nd suppress-ra
end
```

> **i Why RDNSS instead of manual DNS configuration**
>
> Host-B gets its own IPv6 address via SLAAC, not DHCP, so there is no DHCP exchange for it to also learn a DNS server from. RDNSS (RFC 8106) is the IPv6-native answer to that gap; it rides inside the same Router Advertisement that already tells Host-B its prefix, so a host that is entirely SLAAC-configured can still learn a resolver without running DHCPv6. The **no ipv6 nd suppress-ra** command is what makes the CPE send RAs on this interface; without it, none of this reaches Host-B regardless of the RDNSS option being present.

Before restarting anything on Host-B, capture on the CPE interface so you can watch the actual Router Advertisement carrying the new option arrive.

CPE - Linux shell

```
# tcpdump -i eth0 -n -v icmp6

--------------------------- Output omitted ---------------------------

18:20:44.606869 IP6 (flowlabel 0xb1fbc, hlim 255, next-header ICMPv6 (58) payload length: 80)
fe80::a8c1:abff:fe0e:ac7 > ff02::1: [icmp6 sum ok] ICMP6, router advertisement, length 80
        hop limit 64, Flags [none], pref medium, router lifetime 60s, reachable time 0ms,
retrans timer 0ms
          prefix info option (3), length 32 (4): 3fff:100:21::/64, Flags [onlink, auto], valid
time 2592000s, pref. time 604800s
          source link-address option (1), length 8 (1): aa:c1:ab:0e:0a:c7
          rdnss option (25), length 24 (3):  lifetime 60s, addr: 3fff:100:0:e::e
```

You should observe a Router Solicitation from Host-B immediately after the interface comes back up, followed by a Router Advertisement from CPE. Inside that RA, alongside the Prefix Information option you would have seen in earlier activities, there should now be a Recursive DNS Server (RDNSS) option carrying 3fff:**U**00:0:e::e. This is the exact mechanism that populates resolv.conf a moment later, and seeing it on the wire is the clearest possible confirmation that CPE is sending what you configured, independent of whatever Host-B does with it.

```
# more /etc/resolv.conf
```

```
u01-hostB / # more /etc/resolv.conf
# Generated by dhcpcd from eth0.ra
# /etc/resolv.conf.head can replace this line
nameserver 3fff:100:0:e::e
# /etc/resolv.conf.tail can replace this line
u01-hostB / #
```

Step 5.3: Confirm Reachability Before Touching DNS Records

From Host-B, ping WEB directly by its IPv6 address to confirm the data plane is already working. This should already succeed from Activity 4.

```
# ping6 3fff:U00:d:21::b
```

Then try resolving a name that does not exist yet. Both of these should return NXDOMAIN.

HostB

```
# host web.example
# dig AAAA web.example
```

```
u01-hostB / # dig AAAA web.example

; <<>> DiG 9.18.49 <<>> AAAA web.example
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 10042
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 29b0aed0df8b2b1d010000006a986a740b99132dff182d3d (good)
;; QUESTION SECTION:
;web.example.                   IN      AAAA

;; AUTHORITY SECTION:
example.                300     IN      SOA     ns.example. admin.example. 2024090100 7200 3600
1209600 3600

;; Query time: 0 msec
;; SERVER: 3fff:100:0:e::e#53(3fff:100:0:e::e) (UDP)
;; WHEN: Wed Sep 02 18:27:00 UTC 2026
;; MSG SIZE  rcvd: 113

u01-hostB / #
```

> **i Why check reachability before touching DNS at all**
>
> This test proves that your basic network path, the VPN service from Activity 4, works on its own. Testing it now matters because you will shortly see a completely different kind of error: a service name that resolves to an address but will not connect. The only way to tell

whether this is a broken VPN path or a DNS problem is to have proved that the raw IPv6 address works right now.

Step 5.4: Add AAAA Records for WEB and Storage

Before adding AAAA records, we need to check the existing zone file and confirm there are no AAAA records for either service yet on the DNS server.

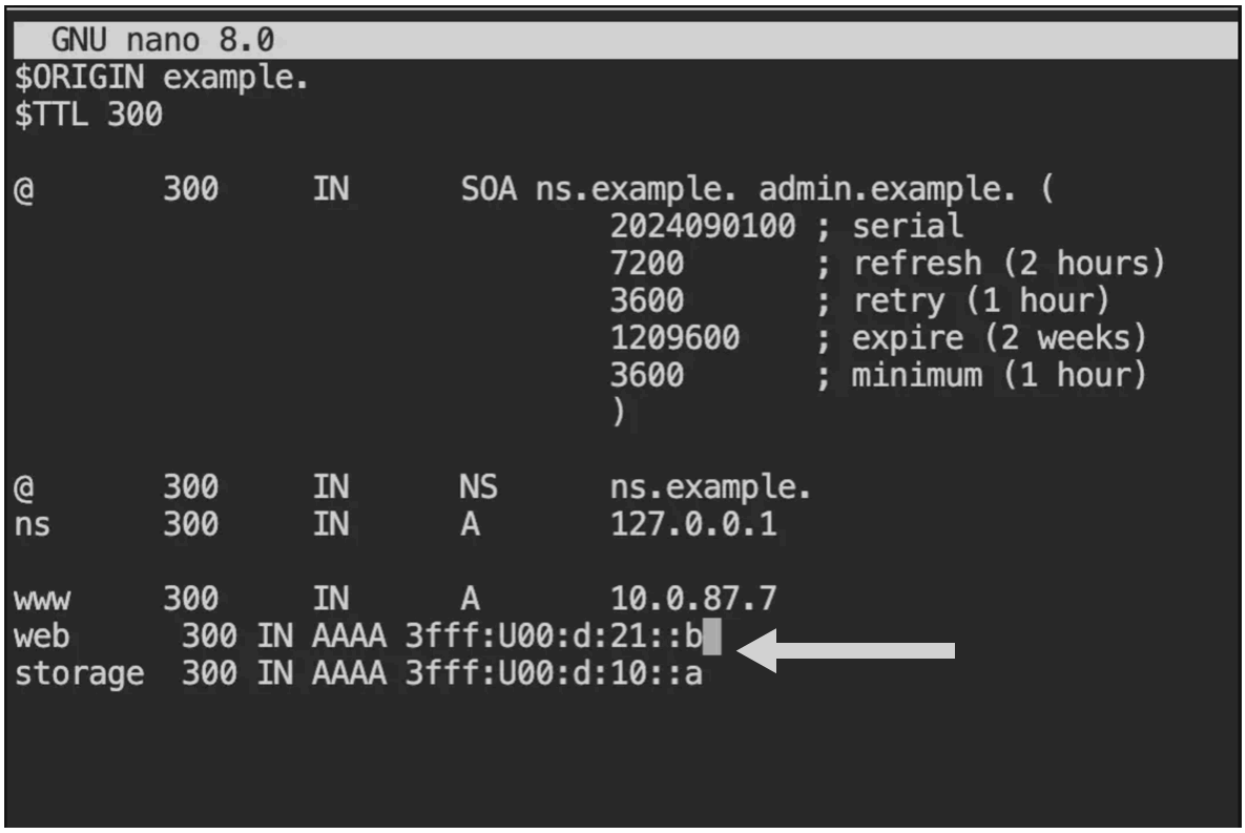
DNS

```
# more /etc/bind/example.zone
```

To edit it you will use the nano text editor.

1. Run: nano /etc/bind/example.zone
2. Use the arrow keys (or Page Down) to move the cursor to the last line of the file.
3. Press End to make sure the cursor is at the end of that line, then press Enter to start a new line.
4. Type the WEB record: web 300 IN AAAA 3fff:**U**00:d:21::b
5. Press Enter again and type the Storage record: storage 300 IN AAAA 3fff:**U**00:d:10::a
6. Press Ctrl+O (Write Out) to save, then press Enter to confirm the filename.
7. Press Ctrl+X to exit nano.
8. Execute rndc reload to refresh the server records.

```
web      300 IN AAAA 3fff:U00:d:21::b
storage  300 IN AAAA 3fff:U00:d:10::a
```



Check the zone file again to confirm both new records are present, then reload the DNS service so the changes take effect.

DNS

```
# more /etc/bind/example.zone        | On the DNS server
# rndc reload                        | On the DNS server
```

```
u01-dns / # rndc reload
server reload successful
u01-dns / #
```

> **i Why both records, on one shared server, for two isolated customers**
>
> DNS resolution and IP reachability are enforced by completely different mechanisms in this lab. DNS is a name-to-address lookup with no concept of VRFs or Route Targets: the DNS server will happily answer a query for storage.example from anyone who asks, because nothing about the DNS protocol knows Customer-1's VPN exists. Whether the resulting address is actually reachable is decided entirely separately, by the BGP VPNv6 import and

export policy you configured in Activity 4. You are about to see both halves of that in Step 5.5.

Step 5.5: Verify From Host-B

Try to resolve the web domain name to its IPv6 address using the "**host**", "**dig**" or "**curl**" commands. In our examples, we use the "**dig**" command. This should now resolve, and the curl should succeed, since WEB is inside Customer-2's own VRF.

HostB

```
# host web.example
# dig AAAA web.example
# curl web.example
```

```
u01-hostB / # dig AAAA web.example

; <<>> DiG 9.18.49 <<>> AAAA web.example
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26973
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 90082e95a9d3d7f5010000006a986d8bf47acfc7d5b981fe (good)
;; QUESTION SECTION:
;web.example.                   IN      AAAA

;; ANSWER SECTION:
web.example.            300     IN      AAAA    3fff:100:d:21::b

;; Query time: 0 msec
;; SERVER: 3fff:100:0:e::e#53(3fff:100:0:e::e) (UDP)
;; WHEN: Wed Sep 02 18:40:11 UTC 2026
;; MSG SIZE  rcvd: 96

u01-hostB / #
```

Now try **Storage** from Host-B using the "**host**", "**dig**" or "**curl**" commands..

HostB

```
# host storage.example
# dig AAAA storage.example
# curl storage.example
```

```
u01-hostB / # dig AAAA storage.example

; <<>> DiG 9.18.49 <<>> AAAA storage.example
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32542
```

```
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 0d6e9b65261208af010000006a986deffd7b81b192936ec8 (good)
;; QUESTION SECTION:
;storage.example.               IN      AAAA

;; ANSWER SECTION:
storage.example.        300     IN      AAAA    3fff:100:d:10::a

;; Query time: 1 msec
;; SERVER: 3fff:100:0:e::e#53(3fff:100:0:e::e) (UDP)
;; WHEN: Wed Sep 02 18:41:51 UTC 2026
;; MSG SIZE  rcvd: 100

u01-hostB / #
```

i Name resolution succeeds, but the connection doesn't

storage.example resolves to a valid AAAA record, because DNS is a shared service and answers queries for both names regardless of who is asking. However, the **curl will time out**, because Host-B is in Customer-2's VRF, which was never given a Route Target import for Customer-1's VPN. This is the same VPN isolation you verified in Activity 4, now visible from the perspective of a real application: DNS resolution and IP reachability are two independent layers, and one working does not guarantee the other will.

Activity 5 Summary

- You configured RDNSS on CPE and confirmed Host-B picked up the shared DNS server via a Router Advertisement.
- You added AAAA records on the DNS server for both Data Centre services.
- You verified that DNS resolution is shared infrastructure, independent of (and not a substitute for) the VPN-level reachability enforced by the Route Targets you set up in Activity 4.