Stackdriver Profiler displays profiling data by using flame graphs. For a conceptual introduction, see <u>Flame graphs</u> (/profiler/docs/concepts-flame).

The flame graph is composed of *frames*. Each frame represents a function in the service:

- The width of a frame corresponds to its consumption of the metric being analyzed. The top frame, for example, represents the entire service and consumes 100% of the metric being analyzed. Therefore, this frame is shown as full width.
- By default, the frame color corresponds, where possible, to the function's package. If package
 information is unavailable, as with Node.js, the names of the source files are used to color the
 function blocks. In a call stack, a change in block color means a transition from one package to
 another. You can change change the way the frames are colored. For more information, see
 <u>Color mode</u> (/profiler/docs/filtering-profiles#color-filter).

The flame graph shows one or more call stacks with the values averaged over a set of profiles:

- Each individual profile represents data collected one time per minute from a single instance of the configured service in a single Compute Engine zone. The collection period for a profile varies with the profile type. See <u>Profile collection</u> (/profiler/docs/concepts-profiling#collection) for more information.
- A maximum of 250 profiles are averaged. If there are more than 250 profiles available, 250 of them are selected randomly as a sample set.
- The top frame, or "root", in a flame graph represents the entire service. Under the "root" frame, is
 another frame or set of frames making up the second row in the graph. Each of these colorcoded frames is a top-level call made by the service. Under each of those colored function
 frames is another set of function frames, each of which is responsible for some part of the
 resource of the frame above it. The hierarchy of function frames in the graph represents the call
 sequence, and the width of a frame represents that function or method's contribution to the
 resource consumption.

For example, the profiler graph of the consumed heap for the docdemo-service shows that the service calls different top-level frames, one of them is the Go runtime's main function. The width of these top-level frames show the majority of the heap is consumed by the call stack involving Go runtime's main, application-specific main, allocOnce, and allocImpl:

Stackdriver Profiler	Timespan 7 days 👻	End time 10/24/19, 7:47 AM E	DT 🛗	DOCUMENTA	TION
Service Profile type Zone docdemo-service ▼ Heap All zones	▼ Version 1.75.0 ▼	Weight All (57.36 MiB - 59.08 MiB), 29 profil ▼	Compare None	to 🗸	0
🗄 🗧 Metric : Bytes 🛞 Add profile data filter				ר	ŧ
57.92 MiB (100%), averaged over 29 profiles					
main					
main					
allocOnce allocImpl					
anochripi					
					- 11

The root frame displays metric consumption information.

When you hold the pointer over a frame in the flame graph, a frame tooltip opens and displays additional information about the frame:

- Function name
- Source file location
- Metric consumption information

In the metric consumption information, the prefix **total** indicates that the metric-value and percentage are for a function and its children. In contrast, the prefix **self** indicates the metric-value and percentage are for the function with the consumption of its children excluded.

The dialog also includes buttons for three filtering actions:

- Focus (/profiler/docs/focusing-profiles)
- Show from frame (/profiler/docs/filtering-profiles#show-from-frame)
- Show stacks (/profiler/docs/filtering-profiles#stack-filters)

In this example, the tooltip shows that the runtime.main function is located in the /usr/local/go/src/runtime/proc.go file. The metric consumption information shows that

runtime.main and its children consume 56.36 MiB of memory, or about 97% of the total memory consumed by the program:

Stackdriver Profiler	Timespan 7 days ▼	NOW End time 10/24/19, 7:47 AM EDT	DOCUMENTATI
Service locdemo-service ▼ Profile type Heap ▼	Zone Version All zones	All (57.36 MiB - 59.08 MiB), 29 profil Compa	re to
E Metric : Bytes 😣 Add profile da	ta filter		ר
7.92 MiB (100%), averaged over 29 profiles ain			
loc0nce locimpl	runtime.main /usr/lib/google-golang/src/runtime/proc.go total: 56.36 MiB, 97% FOCUS SHOW FROM FRAME SHOW STACKS		

When you select a frame, the flame graph is redrawn with that frame shown full width and the colors higher in the call stack are muted. Selecting on a frame doesn't change any settings, it only changes how the graph is displayed. To revert to the original view, select the root (top) frame.

In the following example, the pointer is on a tiny frame and it is difficult to view the call stack. The tooltip identifies (*huffmanBitWriter).write as the frame function:

Stackdriver Profiler			Timespan 7 days NOW			End time 10/24/19, 7:49 AM EDT 🗰 DOC				CUMENTATIO	
Service busybench 🗸 🗸	Profile type CPU time	Zone All zones	✓ Version✓ All versions	▼ Weight All (2.91	s - 3.2	5 s), 10046 profiles ▼	Compare None	e to	•		
II = M	etric : CPU time 🛞 Add j	profile data filter						×	0	1	
.14 s (100%), averag	ged over 250 profiles										
nain.func1									gcBgMa	ark	
usywork									system		
usyworkOnce									gcBgMa		
*Writer).Write						(*Writer).Flush			gcDrain		
*Writer).Write					Ne	(*Writer).Flush			scan		
*compressor).write					n	(*compressor).syncFlush					
*compressor).deflat						(*compressor).deflate				_	
*compressor).write.	(*compresso					(*compress (*		1.			
*huffmanBitWriter)						(*huffmanBi					
*huffmanBitWrit						(*huffma					
*huffmanEn						(*huff					
b ((*						compress/flate.(*huffm	anBitWriter).v	vrite			
Sort (*						/usr/local/go/src/com			t_writer.	:go	
ui S						total: 46.48 ms, 1.48%,	self: 40 µs, 0.0	01%			
q						FOCUS SHOW FRO		SHOW	STACK	/ e	
						FUCUS SHOW FRU		SHOW	STACK	(3	

After you select the frame, the graph is redrawn, making the call stack more visible:

Service Profile type Zone Version Weight Compare to None Image: Weight busybench CPU time All zones All versions All (2.91 s - 3.25 s), 10047 profiles None Image: Compare to the text of text	Frome Bervice Busybench Profile type Zone All zones All versions All versions All (2.91 s - 3.25 s), 10047 profiles None Ompare to None None All zones All versions All (2.91 s - 3.25 s), 10047 profiles None All zones All versions All (2.91 s - 3.25 s), 10047 profiles None All zones All zones All versions All (2.91 s - 3.25 s), 10047 profiles None All zones All zones All zones All zones All versions All (2.91 s - 3.25 s), 10047 profiles None All zones All zone	Stackdriver Profiler	Timespan 7 days	NOW 10/24/		EDT 🛗	DOCUMI	ENTATION
busybench CPU time All zones All versions All (2.91 s - 3.25 s), 10047 profiles None @ Imain funct Imain funct Imain funct Imain funct Imain funct Imain funct busywork Imain funct Imain funct Imain funct Imain funct busywork Imain funct Imain funct Imain funct from pressol syncFlush Imain funct Imain funct (*Writer) Flush Imain funct Imain funct (*Writer) Flush Imain funct Imain funct (*Writer) Flush Imain funct Imain funct (*Mriter) Flush Imain funct Imain funct (*Mriter) Flush Imain funct Imain funct (*Mriter) Flush Imain funct Imain funct (*HuffmanBitWriter) write Imain funct Imain funct (*Buffel) Write Imain funct Imain funct (*Buffel) Grow Imain funct Imain funct Imain funct Imain funct Imain funct<	busybench CPU time All zones All versions All (2.91 s - 3.25 s), 10047 profiles None @ Imain function Imain function Imain function Imain function Imain function Imain function busywork Imain function Imain function Imain function Imain function Imain function busywork Imain function Imain function Imain function Imain function busywork Imain function Imain function Imain function Imain function busywork Imain function Imain function Imain function formeressoril syncFlush Imain function Imain function ("writer) Imain function Imain function ("thuffmanBitWriter) Imain function Imain function ("thuffmanBitWriter) Imain function Imain function ("function Imain function Imain function ("function Imain function Imain function ("thuffmanBitWriter) Imain function Imain function ("thuffmanBitWriter) Imain function Imain function Imain function Imain function Imain function ("function Imain function Imai	V Floillei					are to	
3.15 s (100%), averaged over 250 profiles main.func1 busywork busy	3.15 s (100%), averaged over 250 profiles main.func1 busywork busyworkOnce (*Writer).Flush (*Writer).Flush (*Compressor).syncFlush (*compressor).deflate (*compressor).deflate (*Compressor).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*Buffer).grow makeSlice malkeSlice				s), 10047 profiles 🛛 👻			- 0
main.func1 busywork0nce (*Writer).Flush (*compressor).syncFlush (*compressor).syncFlush (*compressor).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).write (*Buffer).Write (*Duffer).Write (*Duffer)	main.func1 busywork busyworkOnce (*Write?) Flush (*Compressor).syncFlush (*Compressor).syncFlush (*Compressor).syncFlush (*Compressor).syncFlush (*Compressor).syncFlush (*Compressor).syncFlush (*LuffmanBitWrite).writeBlock (*HuffmanBitWrite).writeBlock (*HuffmanBitWrite).writeBlock (*HuffmanBitWrite).writeBlock (*Buffer).grow makeSlice makeSlice makeSlice malloogc systemstack malloogc	E Metric : CPU time (8) Add profile data filter					×	0 ±
(*compressor).syncFlush (*compressor).deflate (*compressor).writeBlock (*uffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*buffer).Write (*Buffer).Write (*Buffer).grow makeSlice makeSlice mallocgo systemstack mallocgo systemstack mallocgo (*mheap).alloc memcIrNoHeapPointers (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap)	(*compressor).syncFlush (*compressor).deflate (*compressor).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWrite).writeBlock (*huffmanBitWrite).writeBlock (*huffmanBitWrite).writeBlock (*Buffer).Write (*Buffer).grow makeSlice makeSlice makeSlice mallocgc systemstack mallocgc.func1 iargeAlloc (*mheap).alloc memcIrNoHeapPointers (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap) (*mheap)	main.func1 busywork busyworkOnce (*Writer).Flush						
(*Buffer).grow compress/flate.(*huffmanBitWriter).write //usr/local/go/src/compress/flate/huffman_bit_writer.go total: 45.6 ms, 1.45% real-out it 45.6 ms, 1.45% total: 45.6 ms, 1.45% real-out it 45.6 ms, 1.45%	(*Buffer).grow compress/flate.(*huffmanBitWriter).write //usr/local/go/src/compress/flate/huffman_bit_writer.go total: 45.6 ms, 1.45% real-out it 45.6 ms, 1.45% total: 45.6 ms, 1.45% real-out it 45.6 ms, 1.45%	(*compressor).syncFlush (*compressor).deflate (*compressor).writeBlock (*huffmanBitWriter).writeBlock (*huffmanBitWriter).writeBytes						
mallocgc profilealloc gc mallocgc.func1 mProf.Malloc gc largeAlloc callers I (*mheap).alloc h systemstack (*mheap).alloc (*mheap) gentraceb (*mheap).alloc (*mheap) gentraceb (*mheap) (*mheap) gentraceb	mallocgc profilealloc gc mallocgc.func1 mProf.Malloc gc largeAlloc callers I (*mheap).alloc h systemstack (*mheap).alloc (*mheap) gentraceb (*mheap).alloc (*mheap) gentraceb (*mheap) (*mheap) gentraceb	(*Buffer).grow makeSlice			compress/flate.(*huffr /usr/local/go/src/com			writer.go
(*mheap).alloc h systemstack memcIrNoHeapPointers (*mheap) callers.fun (*mheap) (*mheap) gentraceb (*mheap) value value	(*mheap).alloc h systemstack memcIrNoHeapPointers (*mheap) callers.fun (*mheap) (*mheap) gentraceb (*mheap) value value	mallocgc systemstack			gc	OM FRAME	SHOW S	TACKS
		(*mheap).alloc	(*mheap)	h systemstack				
					1			

To restore a standard flame graph to the original state, select the root (top) frame. To restore a focused flame graph al state, select the frame that displays the value of the **Focus** filter. For example, if your filter bar contains **Focus: So** the frame with the label **Sort**.

- For information on how to suppress, or highlight, frames in the flame graph, see <u>Using filters</u> (/profiler/docs/filtering-profiles).
- For information on focusing the graph on a single function, see <u>Focusing the graph</u> (/profiler/docs/focusing-profiles).
- For information on comparing profiles collected by different deployments of your service, see <u>Comparing profiles</u> (/profiler/docs/comparing-profiles).
- To download your profile data, see <u>Downloading profiles</u> (/profiler/docs/downloading-profiles).
- For information on using the Profiler agent to collect profiling data for your services, see:
 - <u>Profiling Go applications</u> (/profiler/docs/profiling-go)
 - Profiling Java applications (/profiler/docs/profiling-java)
 - Profiling Node.js applications (/profiler/docs/profiling-nodejs)
 - <u>Profiling Python applications</u> (/profiler/docs/profiling-python)
 - <u>Profiling applications running outside Google Cloud</u> (/profiler/docs/profiling-external)